Université du Québec
École de technologie supérieure

DÉPARTEMENT DE GÉNIE DE LA PRODUCTION AUTOMATISÉE

MICROÉLECTRONIQUE APPLIQUÉE
GPA-770
Cahier de laboratoires

Maurice Tremblay, Éric Granger, Dominique Grenier

Rédaction : MT
Décembre 2004
Révision : décembre 2014

Table des Matières

INTRODUCTION 1

1 LABORATOIRE #1 Programmation assembleur du 68HCS12 3
1.1 LABORATOIRE #1a Mise en place et essai de l'environnement de développement 4
1.1.1 Mise en place de l'environnement de développement 4
1.1.2 Installation d'un émulateur de terminal VT100 (Virtual Terminal 100) 5
1.1.3 Ouverture d'un projet dans CodeWarrior HC(S)12 IDE 9
1.1.3.1 Description des fichiers du projet 15
1.1.3.1.1 Fichier 'Readme.txt' 15
1.1.3.1.2 Fichier 'Main.asm' 16
1.1.3.1.3 Fichier 'burner.bbl' 17
1.1.3.1.4 Fichier 'Simulateur.ini' et fichier 'SofTec.ini' 18
1.1.3.1.5 Fichiers '.cmd' de commandes pour le dévermineur 19
1.1.3.1.6 Fichier 'mc9s12c32.inc' (nomenclature des registres) 22
1.1.3.1.7 Fichier 'D_BUG12M.MAC' 24
1.1.3.1.8 Fichier 'D_BUG12M.ASM' 26
1.1.4 Complément pour votre environnement 30
1.1.4.1 La tabulation 30
1.1.4.2 Le fichier .lst 30
1.1.5 Essai de l'environnement 34
1.1.6 Édition 35
1.1.7 Compilation 38
1.1.8 Téléchargement 40
1.1.8.1 Connexion virtuelle (Simulator) 40
1.1.8.2 Connexion réelle (SofTec) 40
1.1.8.3 Téléchargement vers la connexion choisie 40
1.1.8.4 Déverminage 43
1.1.8.5 Déplacement dans le programme 44
1.2 LABORATOIRE #1b Programme simple avec branchements de base 48
1.2.1 Simulation de la lecture de capteurs infrarouges. 48
1.3 LABORATOIRE #1c Sous-routines et passage de paramètres 49
1.3.1 Passage de paramètres par valeurs et par références 49
1.3.2 Simulation du contrôle de deux moteurs DC 49
1.4 LABORATOIRE #1d Sous-routines d'entrées/sorties D-BUG12M 52
1.5 LABORATOIRE #1e Contrôleur à logique flou avec les instructions du HCS12 54
1.5.1 Contrôle de direction d'un robot 54
1.5.1.1 Nombres de variables requises et attribution des noms 55
1.5.1.2 Développement des règles du contrôleur flou 56
1.5.1.3 Affichage des résultats du contrôleur flou 57
1.6 Rapport pour le laboratoire #1 58

1.6.1 Travail à remettre 58
1.6.2 Barème de correction 58

2 LABORATOIRE #2 Exploitation en temps réel du matériel et des ports d'entrées/sorties 60
2.1 LABORATOIRE #2a Interrogation d'un bouton poussoir. 61
2.1.1 Mode réel 61
2.1.2 Mode simulation 62
2.1.3 Mode simulation et réel 62
2.2 LABORATOIRE #2b E/S, afficheur LCD et interruptions externes 64
2.2.1 Procédure 64
2.2.2 Mode simulation 65
2.2.3 Mode réel 69
2.3 LABORATOIRE #2c Modulation de la durée d'impulsions 71
2.3.1 Introduction 71
2.3.2 Modulation de la durée d'une impulsion 71
2.3.2.1 Module de l'horloge et des minuteurs de temps (TIM) 73
2.3.2.2 Sorties de signaux sur une comparaison d'horloge (OCn) 73
2.3.2.3 Captures des temps d'entrées (ICn). 74
2.3.2.4 Cas particulier avec le OC7 75
2.3.2.5 Profil de vitesse 76
2.3.2.6 Mesure du nombre d'impulsion arrivant aux moteurs 76
2.3.2.7 Routines de service et programme principale 77
2.4 LABORATOIRE #2d Détection du voltage des capteurs infrarouges 78
2.4.1 Le convertisseur A/N du 68HCS12 79
2.4.2 Numérisation des IRs et affichage 81
2.5 Rapport pour le laboratoire #2 83
2.5.1 Travail à remettre 83
2.5.2 Barème de correction 83

3 LABORATOIRE #3 Navigation temps réel du robot mobile 85
3.1 LABORATOIRE #3a Contrôle de la direction du robot mobile 86
3.1.1 Introduction 86
3.1.2 Calcul des braquages 88
3.1.3 Exemple de calcul d'un braquage court 90
3.1.3.1 Choix de la vitesse de braquage 90
3.1.4 Calcul de l'angle de braquage 90
3.2 LABORATOIRE #3b Robot mobile 91
3.2.1.1 Point mort 91
3.2.2 Étalonnage 91
3.2.2.1 Alignement pour la marche avant 91
3.2.3 Suggestions et approches. 91

3.2.3.1 Routine de service d'interruption des moteurs 92
3.2.3.2 Routine sans fin du programme principale 92
3.2.3.3 Routine de service d'interruption pour les pare-chocs 93
3.3 Rapport pour le laboratoire #3 94
3.3.1 Travail à remettre et barème de correction 94

Appendice A 95
Syntaxe de compilation pour le langage micro assembleur de Motorola 95

Appendice B 98
Directives de compilation pour le langage assembleur de Motorola 98

Appendice C 100
Standard du format S_RECORD de MOTOROLA 100

Appendice D 101
Commandes pour le dévermineur 101

Appendice E 105
Table de caractères ASCII 105

Appendice F 106
Circuits électriques de la carte HCS12 STARTER KIT 106

Table des illustrations

Figure 1. Carte HCS12 STARTER KIT de la compagnie SofTec Microsystems. 2
Figure 2. Démarrer/exécuter 5
Figure 3. Description de la connexion 6
Figure 4. Sélection du port de télécommunication 6
Figure 5. Paramètres de télécommunication 7
Figure 6. HyperTerminal 8
Figure 7. Ouverture d'un projet 10
Figure 8. Chemin complet et nom du projet 11
Figure 9. Choix du type de microcontrôleur utilisé 11
Figure 10. Choix du langage de programmation 12
Figure 11. Choix du model de mémoire 12
Figure 12. Choix des connexions pour le dévermineur 13
Figure 13. Sous répertoires créés par le projet1 13
Figure 14. Représentation du projet dans le logiciel CodeWarrior 14
Figure 15. Connexions virtuelles de 8 DELs (**I/O_Led**) et d'un terminal (**Terminal**) 21
Figure 16. Mise en place de la tabulation 30
Figure 17. Création d'un fichier .lst pour l'assembleur en mode connexion au simulateur 31
Figure 18. Création d'un fichier .lst pour l'assembleur en mode connexion à la carte 31
Figure 19. Menu Projet/Compile, /Make et /Debug 38
Figure 20. Compile 38
Figure 21. Make 39
Figure 22. Choix de la connexion pour le dévermineur 39
Figure 23. Debug 40
Figure 24. Erreurs et mises en gardes 41
Figure 25. Dévermineur 43
Figure 26. Interaction du dévermineur 44
Figure 27. Exécuter le code jusqu'à cette ligne 45
Figure 28. Insertion d'un point d'arrêt 46
Figure 29. Changer la valeur du registre d'entrée/sortie PORTB 47
Figure 30. Exemple de la vitesse d'un moteur en mémoire 49
Figure 31. Exemple du profil de la vitesse d'un moteur 50
Figure 32. Localisation des capteurs IRs du robot 54
Figure 33. Définition des fonctions d'appartenance d'entrées pour chacune des variables du contrôleur flou. 56
Figure 34. Définition des fonctions d'appartenance de la sortie du contrôleur flou. 56
Figure 35. Bouton poussoir de la carte d'évaluation 61
Figure 36. Boutons poussoirs en mode simulation 62
Figure 37. Afficheur LCD en mode simulation 65
Figure 38. Registres de contrôle du LCD 66
Figure 39. Instructions possibles au LCD en mode simulation. 67
Figure 40. Étapes d'initialisations du LCD 68
Figure 41. Afficheur LCD DMC 16117A 70
Figure 42. Impulsions envoyées aux contrôleurs des moteurs 72
Figure 43. Contrôleur et moteur CC du robot mobile 72

Figure 44. Potentiometre pour l'étalonnage nul à 1,5ms 73
Figure 45. Contrôle de la vitesse des moteurs gauche et droit par la modulation de la durée 75
Figure 46. Profil de vitesse pour le contrôle des moteurs en marche avant 76
Figure 47. Capteur infrarouge GP2Y0A21YK 78
Figure 48. Voltage analogique en fonction de la distance de réflexion d'un objet 79
Figure 49. braquage court 87
Figure 50. Braquage moyen 87
Figure 51. Braquage long 87
Figure 52. Mesures des roues du robot mobile 89

INTRODUCTION

Afin d'intégrer les concepts de la microélectronique appliquée vus en classe, vous apprendrez à développer les laboratoires proposés dans votre livre de cours **"68HC12 Microcontroller Theory and Applications de Daniel J. Pack et Steven F. Barrett"**. Ces laboratoires ont été ajustés au nouveau microcontrôleur Motorola MC9S12C32 en utilisant l'environnement convivial CodeWarrior IDE (Integreted Development Environment) V 4.5 de la compagnie FreeScale, et adaptés aux robots mobiles développés au département de génie de la production automatisée pour ce cours.

Le microcontrôleur MC9S12C32 a été choisi en fonction de ses nombreuses particularités. Il n'est pas spécialisé à une tâche spécifique et ses nombreuses caractéristiques le rendent des plus versatiles. L'apprentissage de ce microcontrôleur vous permettra de maîtriser facilement l'ensemble des microcontrôleurs que l'on retrouve aujourd'hui sur le marché.

Orchestré par un micro-ordinateur de 16 bits (HCS12), ce microcontrôleur possède une mémoire interne volatile (RAM) de 2048 octets pour stoker les variables ainsi qu'une mémoire interne non volatile (Flash EEPROM) de 32768 octets pour stoker les constantes et le(s) programme(s).

Grâce à ses nombreux modules internes, ce microcontrôleur peut avec le(s):

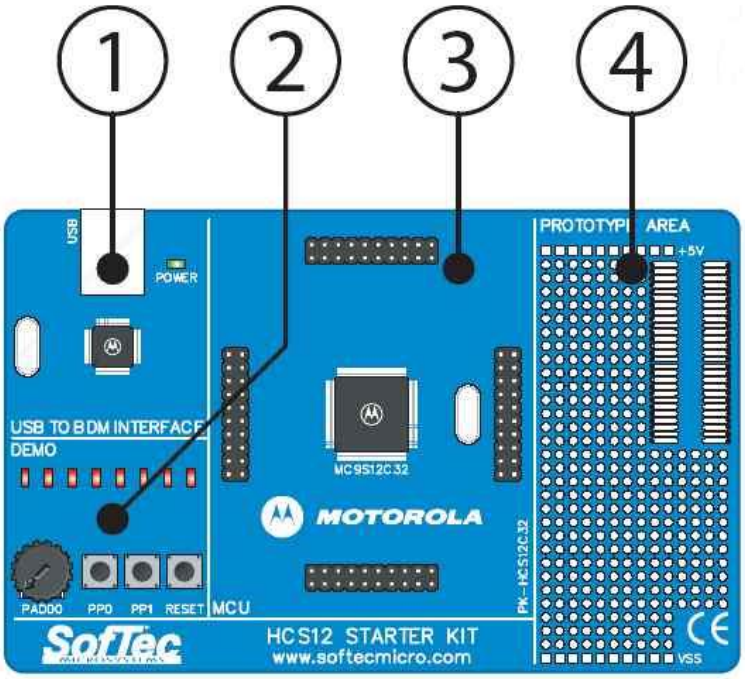
1) **SCI** (**S**erial **C**ommunications **I**nterface) transmettre de façon asynchrone des caractères ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange).
2) **SPI** (**S**erial **P**eripheral **I**nterface) interfacer des périphériques synchrones.
3) **TIM** (**Ti**mer **M**odule) contrôler 8 horloges programmables pour la capture d'entrées ou la comparaison de sorties, un accumulateur d'impulsion de 16 bits, des compteurs d'événements externes, compter le temps entre des événements externes prédéclenchés.
4) **PWM** (**P**ulse **W**idth **M**odulator) contrôler la durée d'une impulsion sur 6 canaux de sorties.
5) **ADC** (**A**nalog to **D**igital **C**onverter) traiter numériquement des signaux analogiques externes.
6) **I/O** (**I**nput/**O**utput) disponibles pour chacun des modules, traiter les informations numériques en entrée comme en sortie.
7) **BDM** (**B**ackground **D**ebugging **M**odule) se mettre en arrière plan et activer son dévermineur interne.

De plus, il est le premier à être doté de fonctions spécialisées à la réalisation de contrôleurs flous.

Les nombreuses composantes du microcontrôleur sont encapsulées dans une seule pastille de 80 broches qui forment le microcontrôleur Motorola MC9S12C32. Vous pouvez localiser ce microcontrôleur sur la carte HCS12 STARTER KIT de la compagnie SofTec Microsystems, dans le laboratoire de microélectronique appliquée (Figure 1). La carte HCS12 STARTER KIT est une carte de développement qui assure une utilisation facile et immédiate du microcontrôleur. Elle

possède un autre microcontrôleur interfaçant le bus USB d'un ordinateur personnel (PC) à son mode BDM. Ainsi interfacé, le bus USB permet l'alimentation de la carte, le téléchargement et le déverminage de programme en temps réel à partir d'un PC. Enfin la carte possède trois commutateurs, huit diodes émettrices de lumière (DELs) et un potentiomètre pour les démos ainsi qu'un espace réservé pour le prototypage d'interfaces électroniques pour le monde extérieur.

Une description plus complète de la carte se trouve dans le fichier **pk-hcs12c32_schematic.pdf** ainsi qu'en appendice.



1- Section USB à BDM
2- Section démonstration
3- Section MCU (**M**icro**c**ontroller **U**nit)
4- Section de prototypage

Figure 1. Carte HCS12 STARTER KIT de la compagnie SofTec Microsystems.

[Ref.: pk-hcs12c32_poster.pdf]

1 LABORATOIRE #1 Programmation assembleur du 68HCS12

Le laboratoire 1 comporte cinq parties qui vous familiariseront avec le microcontrôleur, son environnement, ses différents modes d'adressage et de branchement.

À la fin de ce laboratoire vous serez en mesure de créer, d'éditer, de compiler, de lier et de déverminer un programme assembleur avec les différents outils mis à votre disposition.

1.1 LABORATOIRE #1a Mise en place et essai de l'environnement de développement

1.1.1 Mise en place de l'environnement de développement

1) Installer un émulateur de terminal (clavier et écran) sur votre PC (Personal Computer) qui permettra à votre microcontrôleur de communiquer lexicalement avec un utilisateur.

2) Créer un projet dans l'environnement intégré (IDE, Integreted Development Environment) qui automatisera l'édition, la sauvegarde, la compilation et le déverminage de vos programmes.

1.1.2 Installation d'un émulateur de terminal VT100 (Virtual Terminal 100)

Le microcontrôleur HCS12 peut recevoir et transmettre des caractères ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)[1] via un clavier et un écran grâce à un port sériel programmable spécialisé pour la communication. Dans un premier temps, on vous demande de relier ce port sériel à un terminal, c'est à dire, à un écran et à un clavier.

Un émulateur de terminal (Hyper Terminal) existe dans le système d'exploitation Windows XP. Cet émulateur utilise un port de communication sériel programmable (COM1 à COM8) disponible sur le PC. Une fois l'émulateur de terminal programmé et branché au port de communication sériel du microcontrôleur, vous pourrez recevoir et transmettre des caractères ASCII entre le PC et le microcontrôleur. Des routines de réceptions et de transmissions de caractères ASCII vous seront fournies à cet effet.

Pour se faire, créez une nouvelle connexion en activant le programme **hypertrm.exe** (HyperTerminal) dans **Démarrer/Exécuter**.

Figure 2. Démarrer/exécuter

[1] Voir la table de caractères ASCII (Appendice E)

Une fois activé, dans **Description de la connexion**, donnez un nom à votre connexion, assignez-lui une icône et cliquez **OK**.

Figure 3. Description de la connexion

Une fois fait, la fenêtre **Numéro de téléphone** s'affiche. Sous, **Connecter en utilisant** choisissez le port de communication branché au microcontrôleur soit COM1 à COM8, puis cliquez **OK**.

Figure 4. Sélection du port de télécommunication

Une dernière fenêtre s'affiche pour fixer les **Paramètres** de télécommunication. Sélectionnez les paramètres suivants ; 9600 bits par seconde, 8 bits de données, aucune parité, un bit d'arrêt et aucun contrôle de flux. En cliquant **OK** à cette étape, le port de communication s'active mais votre connexion n'est pas sauvegardée.

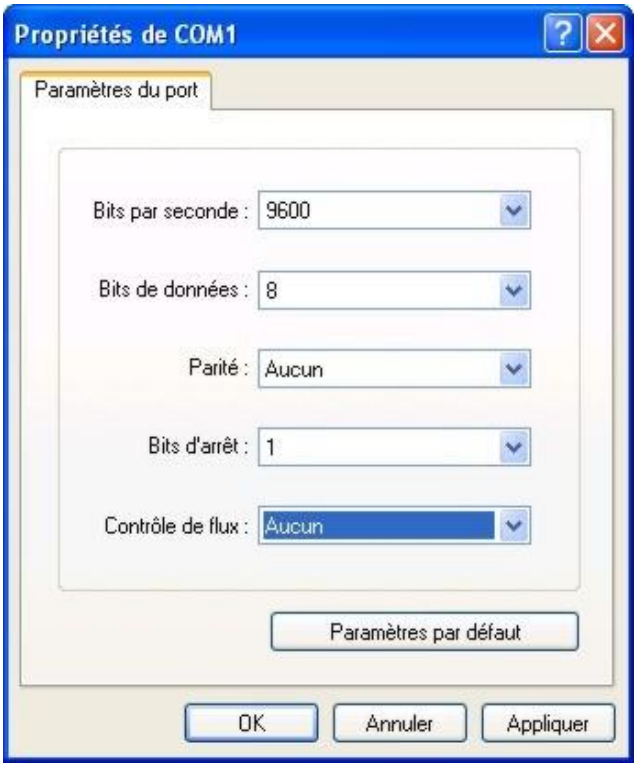


Figure 5. Paramètres de télécommunication

Pour sauvegarder cette connexion dans votre compte[2], allez dans **Fichier** du HyperTerminal, faites **Enregistrer sous**, naviguez jusqu'à votre compte et tapez un nom pour cette connexion.

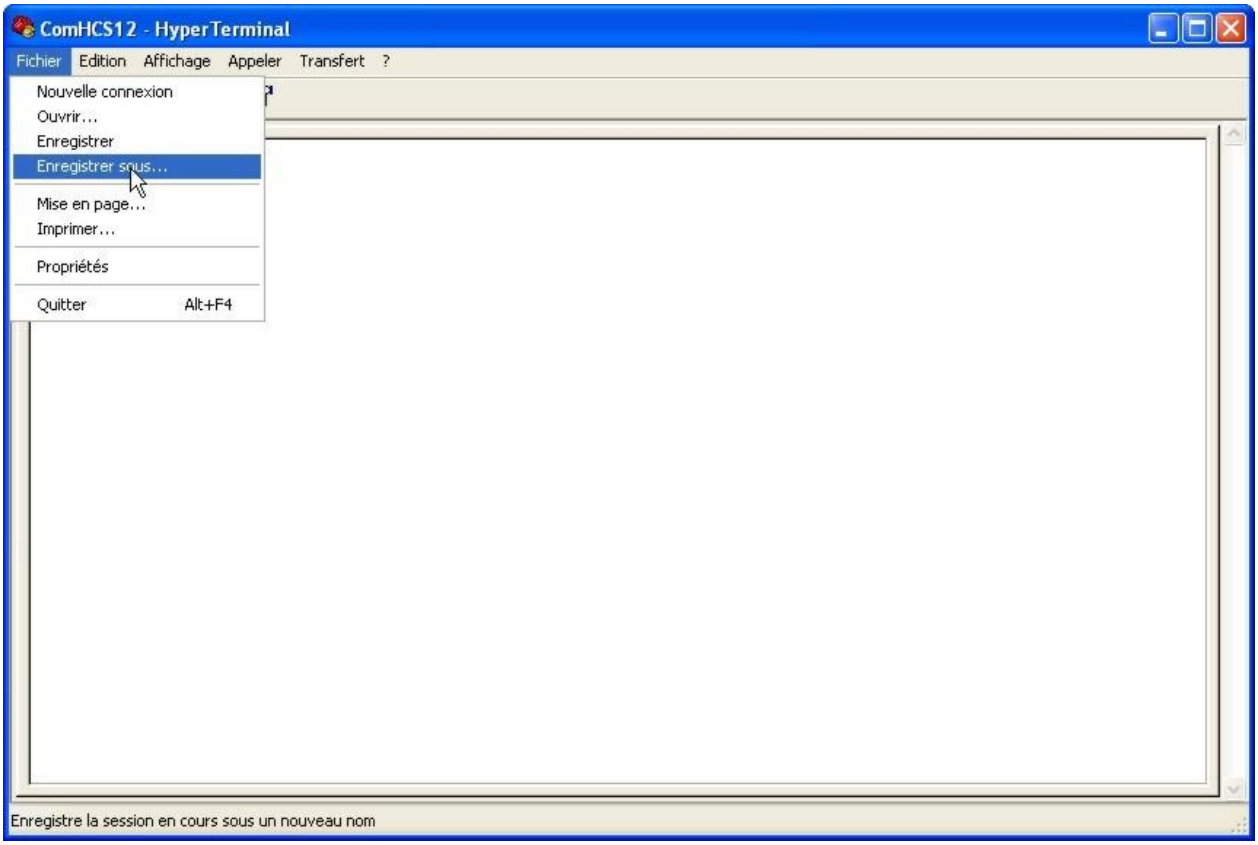


Figure 6. HyperTerminal

Ceci termine la première étape de l'installation de l'environnement.

[2] Un compte est tout simplement un sous répertoire privé dans lequel vous sauvegardez vos programmes, vos projets, vos fichiers de commandes, etc. associé à votre cours.

1.1.3 Ouverture d'un projet dans CodeWarrior HC(S)12 IDE

CodeWarrior, de la compagnie FreeScale, est un logiciel qui intègre un environnement de développement pour plusieurs types de microcontrôleurs dont le 68HCS12 de la compagnie Motorola. On y retrouve un traitement de texte ASCII, différents assembleurs et compilateurs 'C', des générateurs de liens et même un système expert pouvant sous certaines conditions, générer automatiquement une partie du code en langage 'assembleur' ou 'C'.

Cet environnement peut contrôler le BDM (Background Debug Module) du microcontrôleur de la compagnie Motorola par le port USB d'un PC. Le BDM permet le téléchargement et le déverminage en ligne d'un programme résident en mémoire FLASH du microcontrôleur.

De plus, l'environnement de CodeWarrior peut générer un code exécutable que le dévermineur de la compagnie FreeScale peut simuler totalement sur un microcontrôleur virtuel. Ce dévermineur peut même, si on le désire, simuler des composantes électroniques virtuelles que l'on peut brancher au microcontrôleur virtuel. À titre d'exemple, on peut y brancher des DELs, un clavier numérique, un terminal, des commutateurs, etc. **Nul besoin de posséder la carte de développement du microcontrôleur pour faire du pré-prototypage.**

Bien que **'CodeWarrior Development Studio for HC(S)12 Microcontrollers, Special Edition'** soit disponible gratuitement chez www.freescale.com, la version CW12_V4_5.exe disponible sur le site du cours est plus performante.

Fichier: CW12_V4_5.exe 320 791 Ko

L'environnement a besoin d'un grand nombre d'informations pour orchestrer tous vos besoins. Un projet consiste donc à informer l'environnement de développement de vos préférences. On doit y spécifier le langage utilisé, vos préférences d'édition, le type de microcontrôleur utilisé, les différentes commandes de pré et post chargement pour les points de départ du programme et l'installation de composantes électroniques virtuelles, les adresses de téléchargement pour le code et le data, etc..

Pour créer un projet dans l'environnement de CodeWarrior :

- Cliquez sur **File/New**

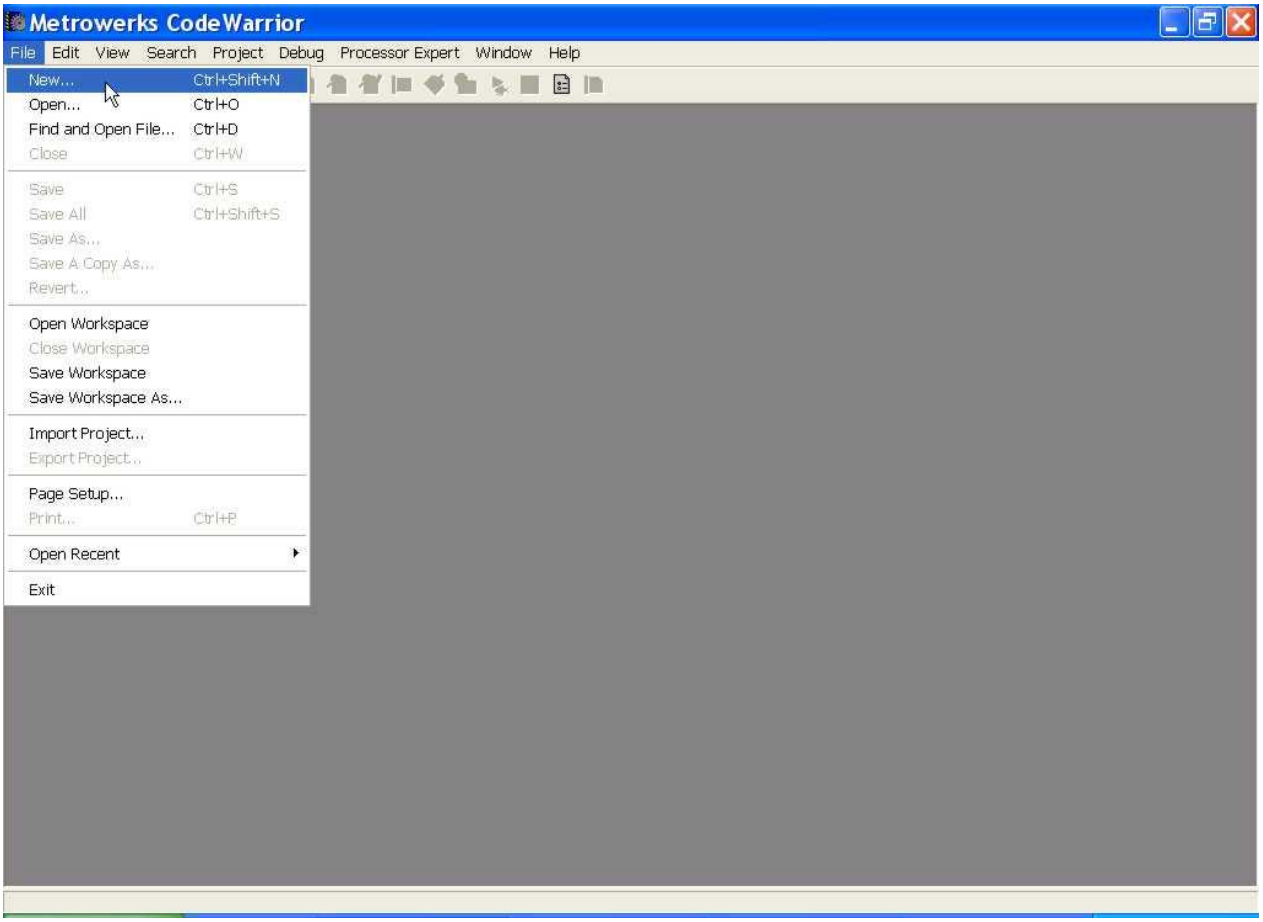


Figure 7. Ouverture d'un projet

- Choisissez l'option **HC(s)12 New Project Wizard**. Donnez un nom à votre nouveau projet dans la ligne Project name, exemple **projet1** et stipuler le chemin d'accès complet de votre sous-répertoire privé dans la ligne 'Location', exemple **F :\GPA770\labo1**, puis cliquez **OK**

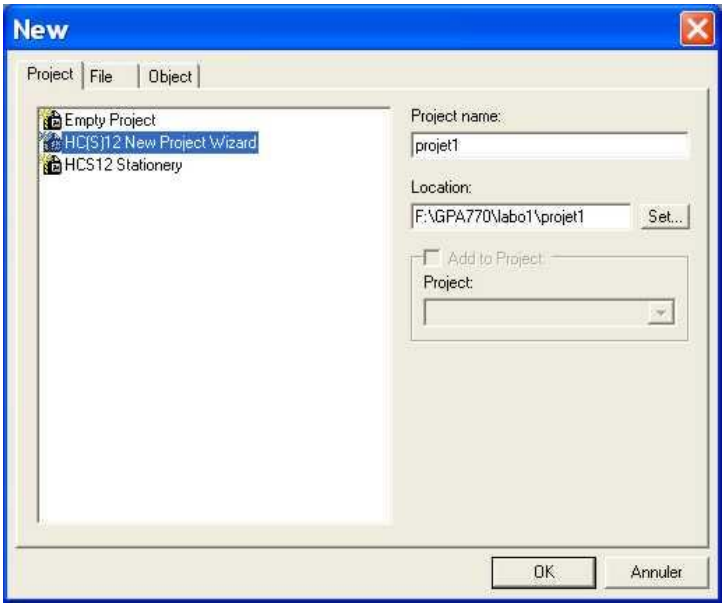


Figure 8. Chemin complet et nom du projet

- Choisissez le microcontrôleur utilisé au laboratoire : MC9S12C32 et cliquez suivant.

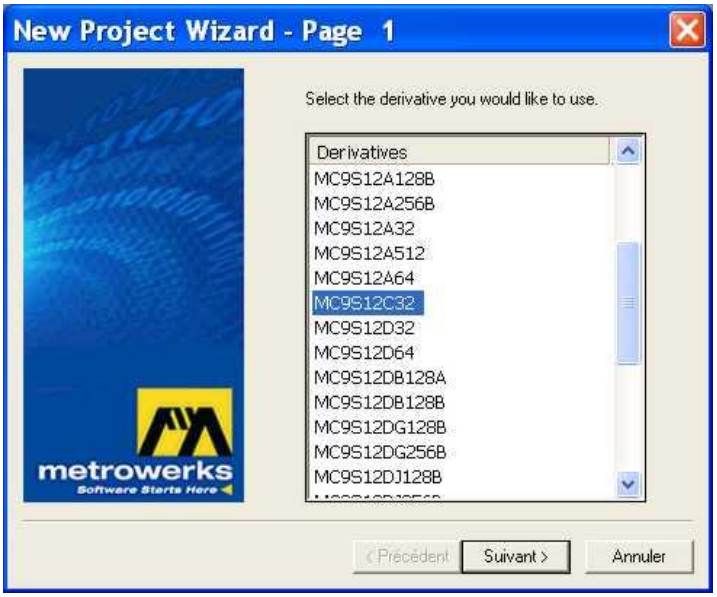


Figure 9. Choix du type de microcontrôleur utilisé

- Sélectionnez le langage assembleur seulement et cliquez suivant.

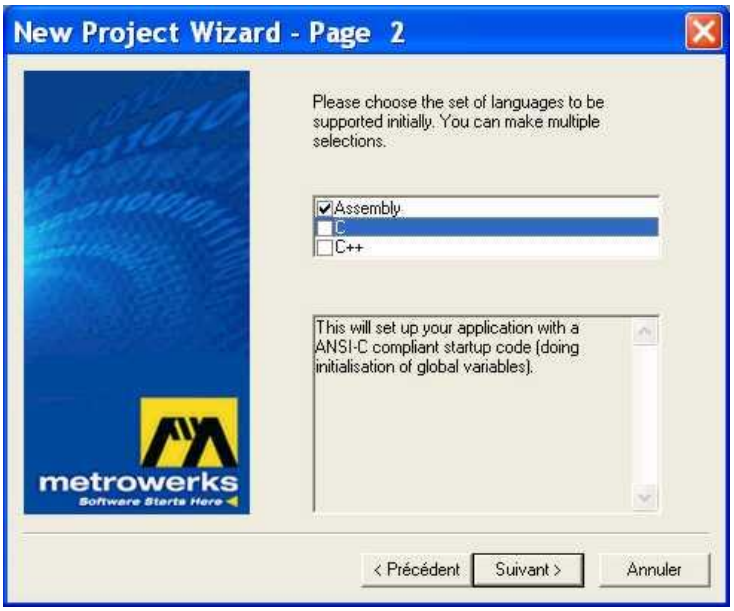

New Project Wizard - Page 2
Please choose the set of languages to be supported initially. You can make multiple selections.
Assembly
C
C++
This will set up your application with a ANSI-C compliant startup code (doing initialisation of global variables).
metrowerks
Software Starts Here
< Précédent
Suivant >
Annuler

Figure 10. Choix du langage de programmation

- Sélectionnez le model d'adresse absolue pour le langage assembleur

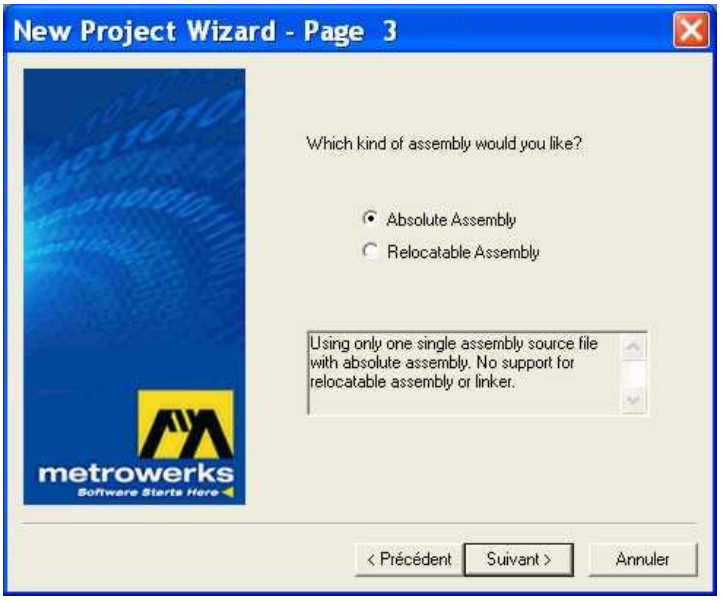

New Project Wizard - Page 3
Which kind of assembly would you like?
Absolute Assembly
Relocatable Assembly
Using only one single assembly source file with absolute assembly. No support for relocatable assembly or linker.
metrowerks
Software Starts Here
< Précédent
Suivant >
Annuler

Figure 11. Choix du model de mémoire

- Sélectionnez les connexions pour le téléchargement et le déverminage. Le **Full Chip Simulator** connectera le devermineur à un microcontrôleur virtuel alors que le **SofTec HCS12** connectera le devermineur à la carte d'évaluation PK-HCS12C32 par le port USB du PC. Cliquez sur terminer quand les choix sont faits.

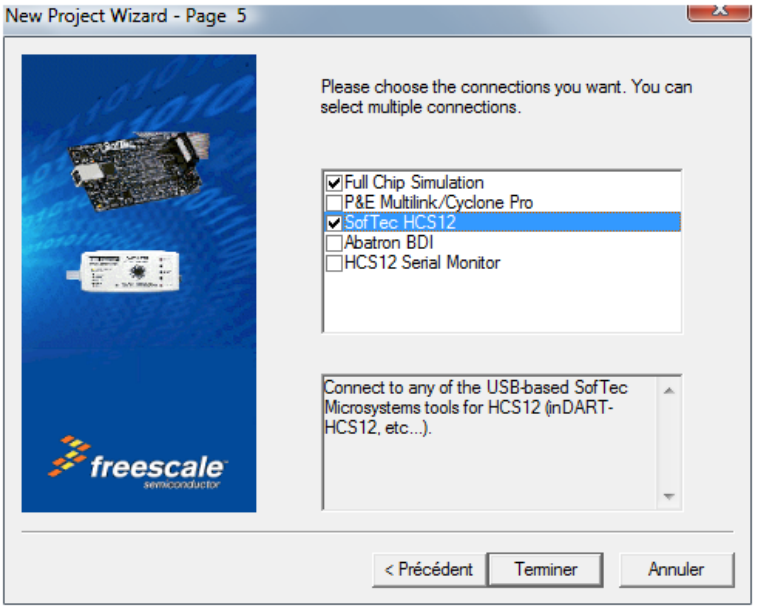

New Project Wizard - Page 5
Please choose the connections you want. You can select multiple connections.
Full Chip Simulation
P&E Multilink/Cyclone Pro
SofTec HCS12
Abatron BDI
HCS12 Serial Monitor
Connect to any of the USB-based SofTec Microsystems tools for HCS12 (inDART-HCS12, etc...).
freescale
< Précédent
Terminer
Annuler

Figure 12. Choix des connexions pour le dévermineur

En cliquant sur **Terminer**, votre projet est créé et sauvegardé dans votre compte. Il est dès lors actif et peut être rappelé par le logiciel CodeWarrior en tout temps en double-cliquant le fichier **projet1.mcp** situé dans le sous-répertoire **projet1** de votre compte.

Le projet a mis en place l'environnement de développement en créant plusieurs fichiers dans des sous répertoires spécialisés dans votre compte comme le montre la figure 14. Pour faire une copie de sécurité de votre projet, vous devrez sauver toute l'arborescence du **projet1**.

Figure 13. Sous répertoires créés par le projet1

Les sous-répertoires ainsi formés par le projet apparaissent dans CodeWarrior comme le montre la figure 15. Il s'agit d'une représentation différente de l'arborescence de Windows. Cette arborescence ne montre que les fichiers texte éditables. En ce qui nous concerne, nous nous concentrerons sur cette représentation simplifiée lors du développement des programmes.

Comme il ne s'agit que de fichier texte, un double click sur un des noms de fichier de cette arborescence ouvre le fichier correspondant dans la partie droite de la fenêtre du logiciel CodeWarrior en mode édition. Ceci implique que l'utilisateur peut en tout temps changer son environnement manuellement.

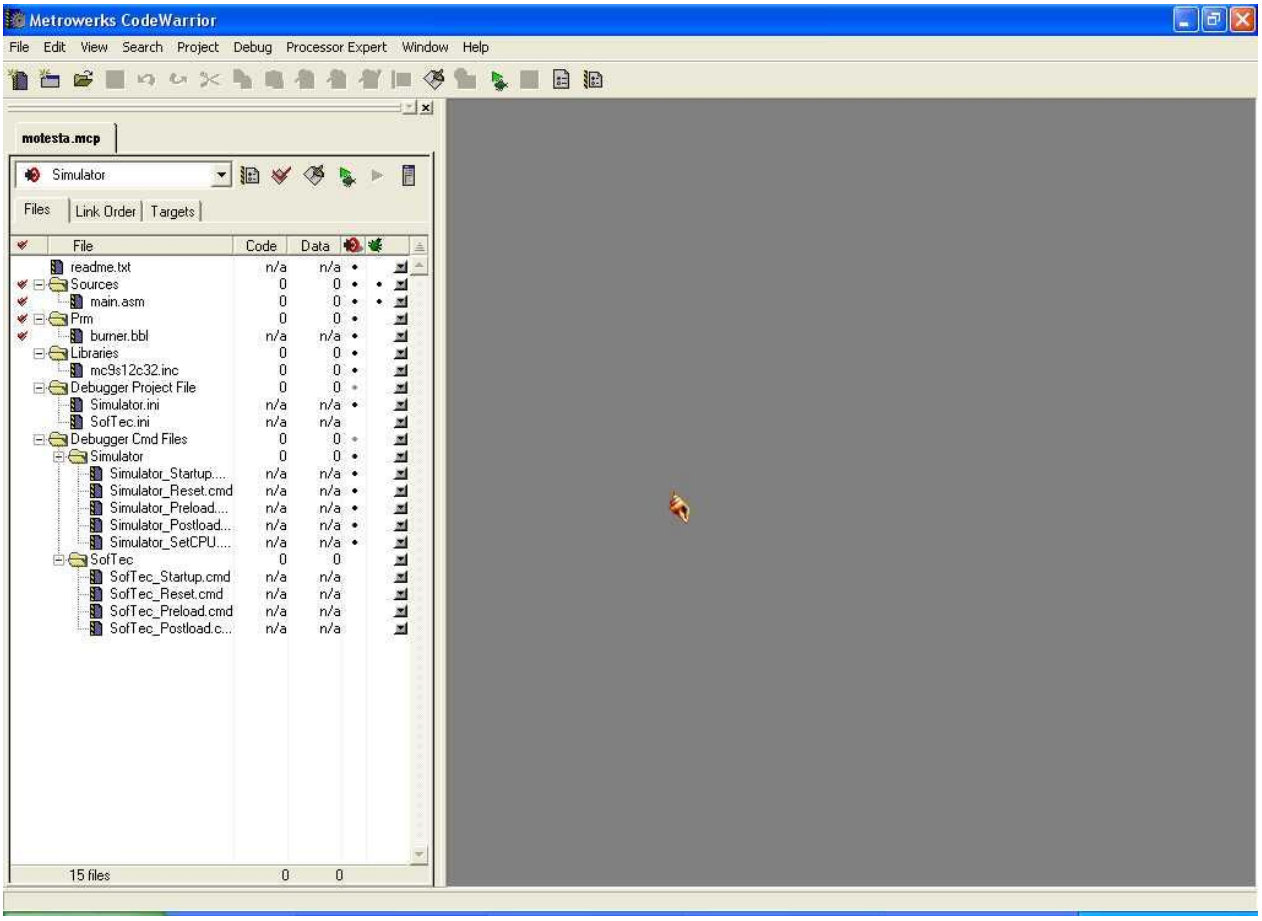


Figure 14. Représentation du projet dans le logiciel CodeWarrior

1.1.3.1 Description des fichiers du projet

1.1.3.1.1 Fichier 'Readme.txt'

Situé dans le sous répertoire **labo1\projej1**, ce fichier contient les paramètres de votre environnement, des conseils d'utilisation pour amorcer votre projet, ainsi qu'une petite définition de la structure de votre projet.

Fichier Readme.txt :

```
//---------------------------------------------------------------------------
//  Tips.txt
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//  Changing the Derivative
//---------------------------------------------------------------------------
If you would like to change the existing derivative with a different one,
you have to consider following points:
- You will need to adapt the CodeWarrior project settings: In the 'Assembler for
  HC12' and 'Compiler for HC12' panels make sure that the new CPU will be used
  in the 'Command Line Arguments':
  - HC12:   -CpuHC12
  - HCS12:  -CpuHCS12
  - HCS12X: -CpuHCS12X
- The derivative header and source files are located in the following directories:
  - {CodeWarrior}lib\hc12c\include (derivative header files)
  - {CodeWarrior}lib\hc12c\src (derivative source files)
  You will need to drag and drop from the directories above the new derivative
  header and source files to the 'Libraries' group.
  HINT: To open a 'Windows Explorer' where the file is located:
        Use the context menu/right mouse click on
        the derivative header/source file in the 'Libraries' project folder
        and select 'Open in Windows Explorer'.

                ...
                ...
                ...
                ...
                ...
                ...
                ...
                ...
                ...
                ...

  - For the HCS12 Serial Monitor Connection:
    Use Processor= HC12 and Connection= HCS12 Serial Monitor
- Optional: In case you need 'Command Files', you can create empty files in the '{Project}cmd'
  directory. In the debugger you can use the 'Command Files...' dialog in the Connection
  menu to adapt the debugger.
- Now you have a new connection in the project.

//---------------------------------------------------------------------------
//  Contacting FreeScale
//---------------------------------------------------------------------------
For bug reports, technical questions, and suggestions, please use the
forms installed in the Release_Notes folder and send them to:
cw_support@freescale.com
```

1.1.3.1.2 Fichier 'Main.asm'

Situé dans le sous répertoire **projet1\Sources**, ce fichier a été créé par votre projet. Il s'agit d'un exemple de programme de départ fonctionnel compatible à l'environnement de développement demandé. Ce code contient la structure de départ minimum pour vos programmes.

Cette structure, à l'aide de directives[3] de compilation (en gras) définie le point d'entrée de votre programme, l'adresse de départ du data, l'adresse de départ du code, génère un petit code test qui sera remplacé par votre propre code et finalement, initialise le vecteur d'interruption de la remise à zéro (RESET) au point d'entrée du programme.

Fichier Main.asm :

```
;*****************************************************************
;* This stationery serves as the framework for a                 *
;* user application (single file, absolute assembly application) *
;* For a more comprehensive program that                         *
;* demonstrates the more advanced functionality of this          *
;* processor, please see the demonstration applications          *
;* located in the examples subdirectory of the                   *
;* Freescale CodeWarrior for the HC12 Program directory          *
;*****************************************************************

; export symbols
            XDEF Entry            ; export 'Entry' symbol
            ABSENTRY Entry        ; for absolute assembly: mark this as application entry point

; include derivative specific macros
            INCLUDE 'mc9s12c32.inc'

ROMStart    EQU  $4000  ; absolute address to place my code/constant data

; variable/data section

            ORG RAMStart
; Insert here your data definition.
Counter     DS.W 1
FiboRes     DS.W 1


; code section
            ORG   ROMStart
Entry:
            LDS   #RAMEnd+1       ; initialize the stack pointer
            CLI                   ; enable interrupts
mainLoop:
            LDX   #1              ; X contains counter
couterLoop:
            STX   Counter         ; update global.
            BSR   CalcFibo
            STD   FiboRes         ; store result
            LDX   Counter
            INX
            CPX   #24             ; larger values cause overflow.
            BNE   couterLoop
            BRA   mainLoop        ; restart.

CalcFibo:  ; Function to calculate fibonacci numbers. Argument is in X.
            LDY   #$00            ; second last
            LDD   #$01            ; last
            DBEQ  X,FiboDone      ; loop once more (if X was 1, were done already)
```

[3] Vous trouverez la définition des directives de compilations en appendice B

```
FiboLoop:
            LEAY  D,Y             ; overwrite second last with new value
            EXG   D,Y             ; exchange them -> order is correct again
            DBNE  X,FiboLoop
FiboDone:
            RTS                   ; result in D


;*************************************************************
;*                 Interrupt Vectors                         *
;*************************************************************
            ORG   $FFFE
            DC.W  Entry           ; Reset Vector
```

1.1.3.1.3 Fichier 'burner.bbl'

Situé dans le sous-répertoire **projet1\Prm**, ce fichier contient les commandes pour le brûleur de EEPROM (FLASH) dans le microcontrôleur. Le brûleur de EEPROM ouvre le fichier '**SofTec_inDART-HCS12.abs.s19'** situé dans le sous-répertoire **bin** en format S-RECORD[4] de Motorola et le télécharge en série par le port USB vers la broche du BDM du microcontrôleur.

Notez que le fichier '**SofTec_inDART-HCS12.abs.s19'** n'existe qu'après avoir compilé votre programme avec succès.

Fichier burner.bbl :

```
/* logical s-record file */
OPENFILE "%ABS_FILE%.s19"
format=motorola
busWidth=1
origin=0
len=0x1000000
destination=0
SRECORD=Sx
SENDBYTE 1 "%ABS_FILE%"
CLOSE

/* physical s-record file */
OPENFILE "%ABS_FILE%.phy"
format = motorola
busWidth = 1
len = 0x4000

origin = 0x008000
destination = 0x000000
SENDBYTE 1 « %ABS_FILE% »
origin = 0x018000
destination = 0x004000
SENDBYTE 1 "%ABS_FILE%"
origin = 0x028000
destination = 0x008000
SENDBYTE 1 "%ABS_FILE%"
.
.
.
origin = 0x3E8000
```

[4] Vous trouverez le format standard S-RECORD de Motorola en appendice C.

```
destination = 0x0F8000
SENDBYTE 1 "%ABS_FILE%"
origin = 0x004000
destination = 0x0F8000
SENDBYTE 1 "%ABS_FILE%"

origin = 0x3F8000
destination = 0x0FC000
SENDBYTE 1 « %ABS_FILE% »
origin = 0x00C000
destination = 0x0FC000
SENDBYTE 1 "%ABS_FILE%"

CLOSE
```

1.1.3.1.4 Fichier 'Simulateur.ini' et fichier 'SofTec.ini'

Vous avez demandé lors de la construction de votre projet, la possibilité d'avoir deux connexions pour votre code. Une connexion vers le simulateur et l'autre vers votre carte PK-HCS12C32. Votre projet a donc créé deux fichiers d'initialisation pour vos deux différents environnements.

Situé dans le sous-répertoire **\projet1**, ces fichiers .ini remplissent les tâches suivantes : 1) pré-conditionnent l'environnement en stipulant les différents chemins d'accès pour les fichiers sources, les fichiers objets, les fichiers exécutables, les bibliothèques et les fichiers d'inclusion, 2) préconditionnent le dévermineur et 3) autorisent l'utilisation des fichiers de commandes que l'on verra plus loin.

Fichier Simulator.ini :

```
[Environment Variables]
GENPATH={Compiler}lib\HC12c\src ;{Compiler}lib\HC12c\include ;{Compiler}lib\HC12c\lib
LIBPATH={Compiler}lib\HC12c\include
OBJPATH={Project}bin
TEXTPATH={Project}bin
ABSPATH={Project}bin

[HI-WAVE]
Target=sim
Layout=ASM_layout.hwl
LoadDialogOptions= AUTOERASEANDFLASH
CPU=HC12

[Simulator]
CMDFILE0=CMDFILE STARTUP ON ".\cmd\simulator_startup.cmd"

[Simulator HC12]
CMDFILE0=CMDFILE RESET ON ".\cmd\simulator_reset.cmd"
CMDFILE1=CMDFILE PRELOAD ON ".\cmd\simulator_preload.cmd"
CMDFILE2=CMDFILE POSTLOAD ON ".\cmd\simulator_postload.cmd"
CMDFILE3=CMDFILE SETCPU ON ".\cmd\simulator_setcpu.cmd"
HCS12_SUPPORT=1
FCS=MC9S12C32
```

Fichier SofTec.ini :

```
[Environment Variables]
GENPATH={Compiler}lib\HC12c\src ;{Compiler}lib\HC12c\include ;{Compiler}lib\HC12c\lib
LIBPATH={Compiler}lib\HC12c\include
```

```
OBJPATH={Project}bin
TEXTPATH={Project}bin
ABSPATH={Project}bin

[HI-WAVE]
Target=gdi
Layout=ASM_layout.hwl
LoadDialogOptions=AUTOERASEANDFLASH
CPU=HC12

[GDI]
COMSETTINGS=SETCOMM DRIVER NOPROTOCOL NOPERIODICAL "SofTec_BDM12.dll"

[SofTec_Microsystems_GdiHCS12]
Hardware=inDART-HCS12
DeviceName=MC9S12C32

[inDART-HCS12_GDI_SETTINGS]
CMDFILE0=CMDFILE STARTUP ON ".\cmd\SofTec_startup.cmd"
CMDFILE1=CMDFILE RESET ON ".\cmd\SofTec_reset.cmd"
CMDFILE2=CMDFILE PRELOAD ON ".\cmd\SofTec_preload.cmd"
CMDFILE3=CMDFILE POSTLOAD ON ".\cmd\SofTec_postload.cmd"
```

1.1.3.1.5 Fichiers '.cmd' de commandes pour le dévermineur

Situés dans le sous-répertoire **projet1\cmd**, ces fichiers sont utilisés pour automatiser les commandes[5] que l'on peut taper manuellement dans le dévermineur selon que la connexion est faite vers le simulateur ou vers la carte PK-HCS12C32

Le simulateur possède 5 fichiers de commandes.

Full_Chip_Simulation_Reset.cmd:
Fichier de commandes exécuté après une remise en route du programme.

Full_Chip_Simulation_Startup.cmd:
Fichier de commandes pour l'amorce du dévermineur. Ce fichier est exécuté après le téléchargement de l'interface de la connexion.

Full_Chip_Simulation_Preload.cmd:
Fichier de commandes exécuté avant le téléchargement du programme.

Full_Chip_Simulation_Postload.cmd:
Fichier de commande exécuté après le téléchargement du programme.

Full_Chip_Simulation_SetCPU.cmd:
Fichier de commandes pour préconditionner le CPU en mode simulation

[5] Vous trouverez les commandes plausibles de ces fichiers en appendice D.

Le SofTec possède quant à lui 4 fichiers de commandes.

SofTec_Reset.cmd:
Fichier de commandes exécuté après une remise en route du programme.

SofTec_Startup.cmd:
Fichier de commandes pour l'amorce du dévermineur. Ce fichier est exécuté après le téléchargement de l'interface de la connexion.

SofTec_Preload.cmd:
Fichier de commandes exécuté avant le téléchargement du programme.

SofTec_Postload.cmd:
Fichier de commande exécuté après le téléchargement du programme..

À titre d'exemple, le programme de bienvenu que vous aurez à écrire, assembler et exécuter un peu plus loin utilisera un terminal **virtuel** pour communiquer lexicalement avec un utilisateur et des DELs **virtuelles** pour afficher un compteur binaire. Pour brancher huit DELs virtuels à un port d'entrée/sortie ainsi qu'un terminal virtuel au microcontrôleur HCS12 en mode simulateur, il faut exécuter les commandes suivantes dans le dévermineur :

OPEN Terminal 40 40 40 40 : Ouverture d'un terminal de communication à la location de l'écran 40x, 40y et de taille 40X40 pixels.

OPEN IO_LED : Ouverture d'un afficheur composé de huit DELs.

PORT 1 : Branchement des DELs sur le PORTB du microcontrôleur à l'adresse 0x0001

DDR 3 : Branchement des DELs au registre de direction (Data Direction Register) du PORTB à l'adresse 0x0003

Pour automatiser ces commandes à chacun des téléchargements de votre programme, il suffit de les placer dans le fichier **Full_Chip_Simulation_SetCPU.cmd** comme suit :

Fichier Full_Chip_Simulation_SetCPU.cmd :

```
// At startup the commands written below will be executed
OPEN Terminal 40 40 40 40
OPEN IO_LED
PORT 1
DDR 3
```

Une fois votre programme écrit, compilé et déverminé, en mode Full_Chip_Simulation, les fenêtres **IO_Led** et **Terminal** s'afficheront et se brancheront automatiquement à votre environnement dès l'amorce de votre programme.

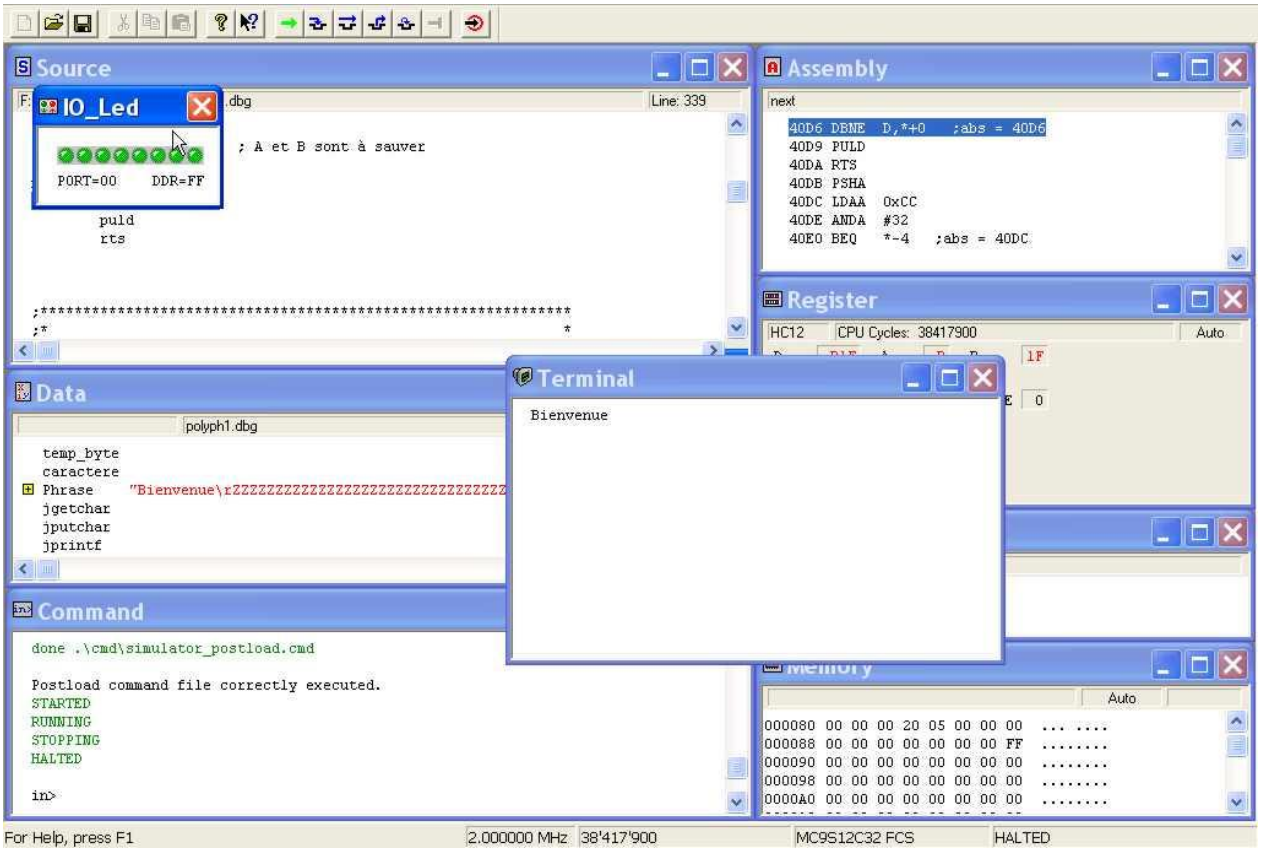


Figure 15. Connexions virtuelles de 8 DELs (**I/O_Led**) et d'un terminal (**Terminal**) au dévermineur en mode simulation

1.1.3.1.6 Fichier ‘mc9s12c32.inc’ (nomenclature des registres)

Le contrôle des modules internes du microcontrôleur se fait à partir d’écritures et de lectures dans des registres appartenant à ces différents modules. Chaque registre est accessible par l’adresse mémoire qui lui est dédiée. Plutôt que de stipuler l’adresse hexadécimale, binaire ou décimale d’un registre pour y accéder, on utilisera de façon ergonomique des noms d’équivalences attachés à ces différentes adresses. Le fichier mc9s12c32.inc est donc un énorme fichier de noms équivalents à des adresses mémoires particulières.

De plus, ce fichier attribut des noms à chacun des bits d’états et de contrôle d’un registre donné.

Exemple :

Vérifier si le convertisseur analogique à numérique a fini une séquence de conversion.

Plutôt que de vérifier le bit 7 de l’adresse $0086, on vérifiera le bit **SCF** (**S**equence **C**omplete **F**lag) de l’adresse **ATDSTAT0** (**A**nalog **T**o **D**igital Converter **Stat**us Register **0**).

Ce fichier doit être inclus en début de chacun de vos programmes avec la directive **INCLUDE** du compilateur assembleur.

Ex. : **INCLUDE** ‘**mc9s12c32.inc**’

Fichier mc9s12c32.inc:

```
; Based on CPU DB MC9S12C32_80, version 2.87.349 (RegistersPrg V2.02)

; ###################################################################
;     Filename  : MC9S12C32.inc
;     Processor : MC9S12C32CFU16
;     FileFormat: V2.02
;     DataSheet : 9S12C-FamilyDGV1/D V01.00
;     Compiler  : CodeWarrior compiler
;     Date/Time : 21.12.2005, 21:43
;     Abstract  :
;         This header implements the mapping of I/O devices.
;
;     (c) Copyright UNIS, spol. s r.o. 1997-2005
;     UNIS, spol. s r.o.
;     Jundrovska 33
;     624 00 Brno
;     Czech Republic
;     http      : www.processorexpert.com
;     mail      : info@processorexpert.com
;
;     File-Format-Revisions:
;      - 14.11.2005, V2.00 :
;        - Deprecated symbols added for backward compatibility (section at the end of this file)
;      - 15.11.2005, V2.01 :
;        - Fixed invalid instruction in macro __RESET_WATCHDOG for HCS12 family.
;      - 17.12.2005, V2.02 :
;        - Arrays (symbols xx_ARR) are defined as pointer to volatile, see issue #2778
;
;     CPU Registers Revisions:
;      - none
```

```
; ###################################################################

;*** Memory Map and Interrupt Vectors
;*************************************
;RAMStart:           equ   $00003800
RAMStart:            equ   $00000800          ; À changer pour l'environnement à la maison
;RAMEnd:             equ   $00003FFF
RAMEnd:              equ   $00000FFF
ROM_C000Start:       equ   $0000C000
ROM_C000End:         equ   $0000FF7F
ROM_4000Start:       equ   $00004000
ROM_4000End:         equ   $00007FFF
;
Vreserved63:         equ   $0000FF80
Vreserved62:         equ   $0000FF82
Vreserved61:         equ   $0000FF84
Vreserved60:         equ   $0000FF86
Vreserved59:         equ   $0000FF88
Vvreglvi:            equ   $0000FF8A
Vpwmesdn:            equ   $0000FF8C
Vportp:              equ   $0000FF8E
Vreserved55:         equ   $0000FF90
Vreserved54:         equ   $0000FF92
Vreserved53:         equ   $0000FF94
Vreserved52:         equ   $0000FF96
Vreserved51:         equ   $0000FF98
Vreserved50:         equ   $0000FF9A
Vreserved49:         equ   $0000FF9C
Vreserved48:         equ   $0000FF9E
Vreserved47:         equ   $0000FFA0
Vreserved46:         equ   $0000FFA2
Vreserved45:         equ   $0000FFA4
Vreserved44:         equ   $0000FFA6
Vreserved43:         equ   $0000FFA8
Vreserved42:         equ   $0000FFAA
Vreserved41:         equ   $0000FFAC
Vreserved40:         equ   $0000FFAE
Vcantx:              equ   $0000FFB0
Vcanrx:              equ   $0000FFB2
Vcanerr:             equ   $0000FFB4
Vcanwkup:            equ   $0000FFB6
Vflash:              equ   $0000FFB8
Vreserved34:         equ   $0000FFBA
Vreserved33:         equ   $0000FFBC
Vreserved32:         equ   $0000FFBE
Vreserved31:         equ   $0000FFC0
.
.
.
;*** ATDSTAT0 - A/D Status Register 0; 0x00000086 ***
ATDSTAT0:            equ    $00000086      ;*** ATDSTAT0 - A/D Status Register 0; 0x00000086 ***
; bit numbers for user in BCLR, BSET, BRCLR and BRSET
ATDSTAT0_CC0:        equ    0                                    ; Conversion Counter 0
ATDSTAT0_CC1:        equ    1                                    ; Conversion Counter 1
ATDSTAT0_CC2:        equ    2                                    ; Conversion Counter 2
ATDSTAT0_FIFOR:      equ    4                                    ; FIFO Over Run Flag
ATDSTAT0_ETORF:      equ    5                                    ; External Trigger Overrun Fla
ATDSTAT0_SCF:        equ    7                                    ; Sequence Complete Flag
; bit position masks
mATDSTAT0_CC0:       equ    %00000001                            ; Conversion Counter 0
mATDSTAT0_CC1:       equ    %00000010                            ; Conversion Counter 1
mATDSTAT0_CC2:       equ    %00000100                            ; Conversion Counter 2
mATDSTAT0_FIFOR:     equ    %00010000                            ; FIFO Over Run Flag
mATDSTAT0_ETORF:     equ    %00100000                            ; External Trigger Overrun Flag
mATDSTAT0_SCF:       equ    %10000000                            ; Sequence Complete Flag
.
.
.
```

1.1.3.1.7 Fichier ‘D_BUG12M.MAC’

Ce fichier contient les définitions des macros pour des appels simples en assembleur des fonctions du simulateur de D_BUG12 de Motorola.

Fonctions :

getchar		Retourne un caractère tapé sur le clavier dans le registre B.
putchar		Transmet le caractère contenu dans le registre B vers le terminal.
out2hex	**arg**	Traduit les deux quartets hexadécimal de l’argument en code ASCII et les transmet au terminal.
out4hex	**arg**	Traduit les quatre quartets hexadécimale en code ASCII et les transmet au terminal.
printf	**arg**	Transmet la phrase débutant à l’adresse de l’argument jusqu’à la rencontre du caractère terminateur de phrase ‘$00’.

N.B. arg = un des modes d’adressage du 68HC12

Ce fichier doit être inclus en début de chacun de vos programmes avec la directive INCLUDE du compilateur assembleur.

Ex. : **INCLUDE** ‘D_BUG12M.MAC’

Ce fichier ne fait pas partie de votre projet. Vous devez le copier dans votre sous-répertoire projet1\Sources.

```
;*********************************************************************
;*                                                                   *
;*  Définition de macros pour des appels simples en assembleur       *
;*                                                                   *
;*  Auteur : Maurice Tremblay                                        *
;*  Date :   août 2004                                               *
;*                                                                   *
;*                                                                   *
;*  Fontions: getchar, putchar, out2hex, out4hex, printf             *
;*                                                                   *
;*********************************************************************


getchar:    macro
            jsr     [jgetchar,pcr]  ; retourne le caractère dans le registre b
            endm

putchar:    macro                   ; l’argument est le caractère ascii à transmettre
                                    ; chargé dans b
            jsr     [jputchar,pcr]  ; le caractère transmis est retourné dans b
            endm

out2hex:    macro                   ; l’argument est un octet en hex
            ldab    \1              ; chargé l’argument dans b
            jsr     [jout2hex,pcr]  ; envoyer l’octet en ascii au terminal
```

```
            endm

out4hex:    macro                   ; l'argument est un mot en hex
            ldd     \1              ; chargé l'argument dans bd
            jsr     [jout4hex,pcr]  ; envoyer le mot en ascii au terminal
            endm

printf:     macro                   ; l'argument est l'adresse de départ
            ldd     \1              ; chargé l'argument dans d
            jsr     [jprintf,pcr]   ; envoyer la phrase au terminal
            endm
```

1.1.3.1.8 Fichier 'D_BUG12M.ASM'

Fichier d'émulation maison de quelques-unes des fonctions de D_BUG12 de Motorola en se servant de la table de vecteur du dévermineur.

La table de vecteur contient à des adresses spécifiques, l'adresse de départ des fonctions programmées : Getchar, Putchar, Out2hex, Out4hex et Printf

Ce fichier doit être inclus **à la toute fin de votre code** mais avant la directive ORG $FFFE , de chacun de vos programmes avec la directive INCLUDE du compilateur assembleur.

Ex. :

```
        INCLUDE  'D_BUG12M.ASM'
;*****************************************************************
;*                  Interrupt Vectors                            *
;*****************************************************************
            ORG   $FFFE
            DC.W  Entry           ; Reset Vector
            END                     ; fin de compilation
```

Ce fichier ne fait pas partie de votre projet. Vous devez le copier dans votre sous-répertoire projet1\Sources.

```
;*****************************************************************
;*                                                               *
;*  Routines de simulation de D-BUG12                            *
;*                                                               *
;*  Auteur : Maurice Tremblay                                    *
;*  Date :   août 2004                                           *
;*                                                               *
;*                                                               *
;*****************************************************************

;*****************************************************************
;*                                                               *
;*  Routines : jgetchar                                          *
;*  Entrée :   aucune                                            *
;*  Tâche :    lire un caractère en provenance du terminal       *
;*  Sortie :   caractère ascii dans le registre b                *
;*                                                               *
;*****************************************************************

Getchar:
        psha
Getchar1:
        ldaa  SCISR1
        anda  #$20      ; RDRF = 1 ?
        beq   Getchar1

        ldab  SCIDRL    ; lecture du caractère
        pula
        rts
```

```
;*************************************************************
;*                                                           *
;*  Routines : jputchar                                      *
;*  Entrée :   registre b contient le caratère ascii         *
;*  Tâche :    transmettre à un terminal un caractère ascii  *
;*  Sortie :   caractère ascii dans le registre b            *
;*                                                           *
;*************************************************************

Putchar:
        psha
Putchar1:
        ldaa  SCISR1
        anda  #$40        ; TDRE = 1 ?
        BEQ   Putchar1

        clr   SCIDRH
        stab  SCIDRL      ; écriture du caractère
        pula
        rts

;*************************************************************
;*                                                           *
;*  Routines : jout2hex                                      *
;*  Entrée :   registre b contient le code hexadécimal       *
;*  Tâche :    transmettre à un terminal la valeur ascii     *
;*             du nombre hexadécimal                         *
;*  Sortie :   aucune                                        *
;*                                                           *
;*************************************************************

Out2hex:
        pshb            ; sauvegarder b
        lsrb
        lsrb
        lsrb
        lsrb
        cmpb  #$0a
        bhs   lettre1   ; si plus grand ou égal c'est + 37h

chiffre1:
        addb  #$30      ; traduire le quartet en ascii
        bra   suitehex1

lettre1:
        addb  #$37      ; traduire le quartet en ascii

suitehex1:
        putchar         ; envoyer premier caractère

        pulb            ; on reprend b
        andb  #$0f      ; garger la partie basse
        cmpb  #$0a
        bhs   lettre2   ; si plus grand ou égal c'est + 37h

chiffre2:
        addb  #$30      ; traduire le quartet en ascii
        bra   suitehex2

lettre2:
        addb  #$37      ; traduire le quartet en ascii
```

```
suitehex2:
        putchar         ; envoyer premier caractère

        rts

;*************************************************************
;*                                                           *
;*  Routines : jout4hex                                      *
;*  Entrée :   registre d contient le code hexadécimal       *
;*  Tâche :    transmettre à un terminal la valeur ascii     *
;*            du nombre hexadécimal                          *
;*  Sortie :   aucune                                        *
;*                                                           *
;*************************************************************

Out4hex:
        pshd                    ; sauvegarde de d
        tab                     ; faire partie haute
        jsr  Out2hex

        puld                     ; recupérer partie basse
        jsr  Out2hex             ; faire partie basse
        rts


;*************************************************************
;*                                                           *
;*  Routines : jprintf                                       *
;*  Entrée :   registre d contient l'adresse de départ       *
;*  Tâche :    transmettre à un terminal un phrase de texte  *
;*            ascii terminée par un 00hex                    *
;*  Sortie :   aucune                                        *
;*                                                           *
;*************************************************************

Printf: pshx                    ; sauvegarde de X et B
        xgdx

DebutPrintf:

        ldab 0,x
        beq   FinPrintf
        putchar
        inx
        bra DebutPrintf

FinPrintf:
        pulx                    ; restorer B et X
        rts

;*************************************************************
;*                                                           *
;*  Chargement de la table de vecteur pour la simulation     *
;*  de D_BUG12                                               *
;*                                                           *
;*                                                           *
;*************************************************************

        ORG   $FE02

jgetchar:  dc.w  Getchar
jputchar:  dc.w  Putchar
```

```
jprintf:   dc.w  Printf

        ORG   $FE16

jout2hex:  dc.w  Out2hex
jout4hex:  dc.w  Out4hex
```

1.1.4 Complément pour votre environnement

1.1.4.1 La tabulation

Positionnez la tabulation à quatre espaces pour le langage assembleur.

- Dans CodeWarrior, sélectionnez **Edit/Preference**, dans la fenêtre **IDE Preferences** cliquez sur **Font & Tabs** et tapez 4 dans **Tab size.**

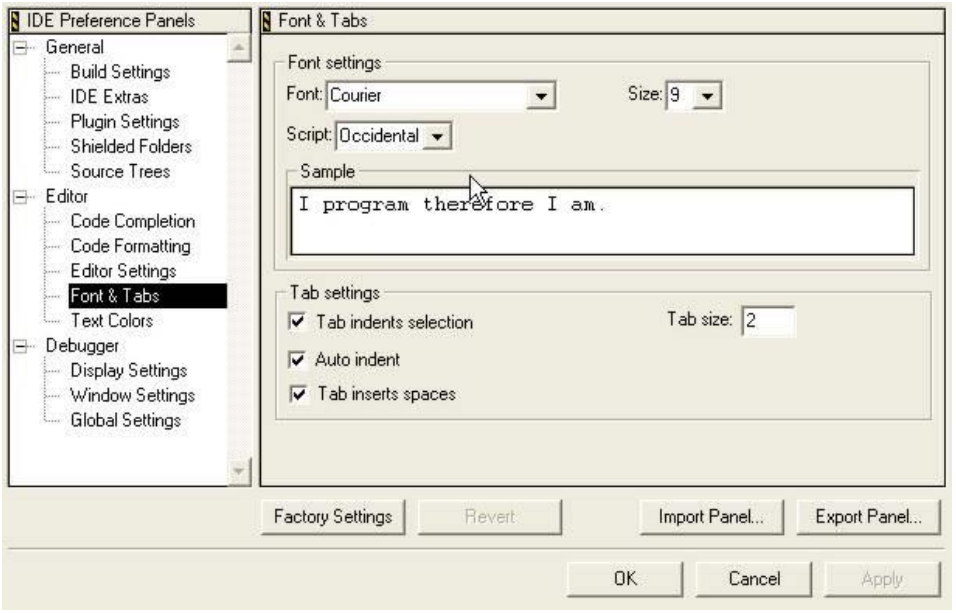


Figure 16. Mise en place de la tabulation

1.1.4.2 Le fichier .lst

Après la compilation de votre code, le fichier .LST contiendra à la fois 1) les numéros de lignes relatives de vos fichiers d'inclusion et vos macros (INCLUDEs et MACROs), 2) les numéros de lignes absolues de votre programme, 3) les adresses ainsi que le code objet qui y est contenu, et enfin 4) votre programme source.

Pour ce faire, vous devez stipuler à votre environnement de générer, pour vos deux connections, un fichier .lst à partir de votre code.

- Pour une connexion en mode simulation, dans CodeWarrior sélectionnez **Project\Set Default Target\Simulator** puis **Edit\Simulator Setting…** Dans la fenêtre Simulator Setting, cliquez sur **Assembler for HC12.** Cliquez sur le bouton Option puis sélectionnez **Generate a listing file**

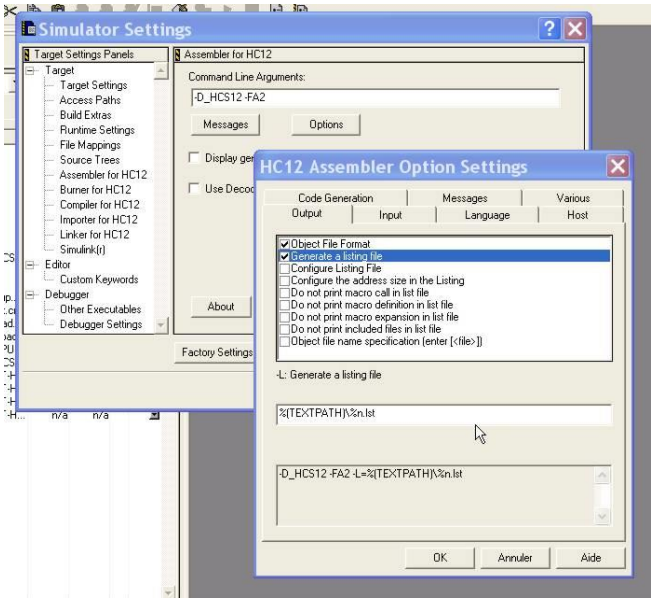


Figure 17. Création d'un fichier .lst pour l'assembleur en mode connexion au simulateur

- Pour une connexion à la carte PK_HCS12C32 en mode SofTec_inDART-HCS12, dans CodeWarrior sélectionnez **Project\Set Default Target\ SofTec** puis **Edit\ SofTec Setting…** Dans la fenêtre Simulator Setting, cliquez sur **Assembler for HC12.** Cliquez sur le bouton Option puis sélectionnez **Generate a listing file**

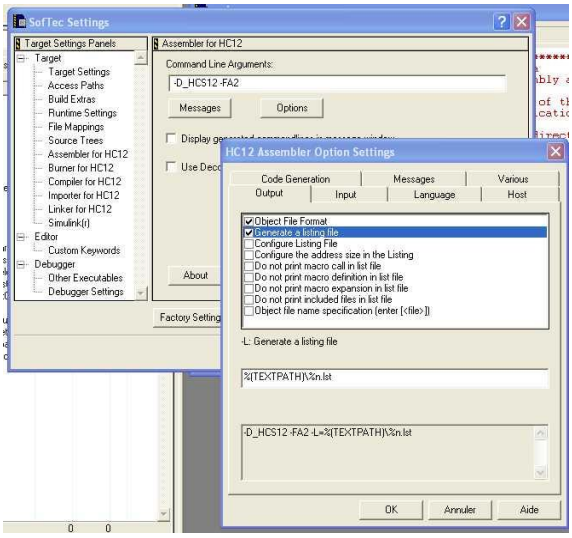


Figure 18. Création d'un fichier .lst pour l'assembleur en mode connexion à la carte PK_HCS12C32

Le fichier .lst sera alors créé **(après la compilation)** dans le sous-répertoire **projet1\bin**. Pour qu'il apparaisse dans l'arborescence de CodeWarrior, cliquez avec le bouton droit de la souris sur le sous-répertoire **Sources**. Dans le pop-up, cliquez sur **Add Files…** et sélectionnez le fichier .lst.

Exemple d'un fichier main.lst après compilation sans erreur :

```
Freescale HC12-Assembler
(c) Copyright Freescale 1987-2005

 Abs. Rel.   Loc    Obj. code   Source line
 ---- ----   ------ ---------   -----------
    1    1                     ;*****************************************************************
    2    2                     ;* This stationery serves as the framework for a                 *
    3    3                     ;* user application (single file, absolute assembly application) *
    4    4                     ;* For a more comprehensive program that                         *
    5    5                     ;* demonstrates the more advanced functionality of this          *
    6    6                     ;* processor, please see the demonstration applications          *
    7    7                     ;* located in the examples subdirectory of the                   *
    8    8                     ;* Freescale CodeWarrior for the HC12 Program directory          *
    9    9                     ;*****************************************************************
   10   10
   11   11                      ; export symbols
   12   12                      XDEF Entry              ; export 'Entry' symbol
   13   13                      ABSENTRY Entry          ; for absolute assembly: mark this as
                                                        ; application entry point
 5399   18          0000 4000   ROMStart    EQU  $4000  ; absolute address to place my
                                                        ; code/constant data
 5400   19
 5401   20                      ; variable/data section
 5402   21
 5403   22                                  ORG RAMStart
 5404   23                       ; Insert here your data definition.
 5405   24  a003800             Counter     DS.W 1
 5406   25  a003802             FiboRes     DS.W 1
 5407   26
 5408   27
 5409   28                      ; code section
 5410   29                                  ORG   ROMStart
 5411   30                      Entry:
 5412   31  a004000 CF40 00                 LDS   #RAMEnd+1       ; initialize the stack pointer
 5413   32  a004003 10EF                   CLI                   ; enable interrupts
 5414   33                      mainLoop:
 5415   34  a004005 CE00 01                 LDX   #1              ; X contains counter
 5416   35                      couterLoop:
 5417   36  a004008 7E38 00                 STX   Counter         ; update global.
 5418   37  a00400B 070E                    BSR   CalcFibo
 5419   38  a00400D 7C38 02                 STD   FiboRes         ; store result
 5420   39  a004010 FE38 00                 LDX   Counter
 5421   40  a004013 08                     INX
 5422   41  a004014 8E00 18                 CPX   #24             ; larger values cause overflow.
 5423   42  a004017 26EF                   BNE   couterLoop
 5424   43  a004019 20EA                    BRA   mainLoop        ; restart.
 5425   44                      ; Function to calculate fibonacci numbers. Argument is in X.
 5426   45                      CalcFibo:
 5427   46  a00401B CD00 00                 LDY   #$00            ; second last
 5428   47  a00401E CC00 01                 LDD   #$01            ; last
 5429   48  a004021 0405 07                 DBEQ  X,FiboDone      ; loop once more (if X was 1,
                                                                   ; were done already)
 5430   49                      FiboLoop:
 5431   50  a004024 19EE                    LEAY  D,Y             ; overwrite second last with
                                                                   ; new value
 5432   51  a004026 B7C6                    EXG   D,Y             ; exchange them -> order is
                                                                   ; correct again
 5433   52  a004028 0435 F9                 DBNE  X,FiboLoop
 5434   53                      FiboDone:
 5435   54  a00402B 3D                      RTS                   ; result in D
 5436   55
```

```
5437   56
5438   57                      ;************************************************************
5439   58                      ;*                 Interrupt Vectors                        *
5440   59                      ;************************************************************
5441   60                                  ORG   $FFFE
5442   61  a00FFFE 4000                   DC.W  Entry           ; Reset Vector
```

Notez que le contenu de ce type de fichier sera expliqué en détail un peu plus loin.

1.1.5 Essai de l'environnement

Tout est en place pour éditer, compiler, télécharger et déverminer un programme assembleur pour le microcontrôleur MC9S12C32. Vous allez donc tester cet environnement à partir d'un programme de bienvenue qui vous est fourni un peu plus loin à titre d'exemple.

Ce programme a les propriétés suivantes :

1) affiche un message de bienvenue sur l'émulateur de terminal VT100 via une connexion sérielle RS232 en utilisant le module de télécommunication SCI (**S**erial **C**ommunication **I**nterface).
2) **accepte des caractères à partir d'un clavier et en fait l'écho sur le terminal jusqu'à ce que le caractère [ENTER] soit tapé par l'utilisateur**
3) traduit des chiffres hexadécimal en caractère ascii pour les afficher sur le terminal.
4) active un compteur binaire et affiche ce compteur sur les huit DELs branchées au port d'entrée/sortie B du microcontrôleur.

Vous apprendrez bientôt, avec le langage assembleur, comment programmer et utiliser les différents modules comme le SCI, ainsi que les différents ports d'entrées/sorties du microcontrôleur comme le port B.

À cette étape, il n'est pas nécessaire de comprendre comment fonctionne ce programme.

1.1.6 Édition

Pour un premier contact avec l'éditeur, sauvegardez le fichier MAIN.ASM en LAB1A.ASM et modifiez-le comme dans l'exemple suivant :

Exemple d'un programme classique de bienvenu en assembleur.

```
;*****************************************************************************
;*                                                                           *
;*  Auteur : Maurice Tremblay                                                *
;*  Date :   avril 2007                                                      *
;*                                                                           *
;*                                                                           *
;*  Ce programme :                                                           *
;*                                                                           *
;*  1)                                                                       *
;*  affiche un message de bienvenue par une transmission                     *
;*  sérielle de caractères du microcontrôleur, port SCI, vers le port de     *
;*  communication COM1 d'un ordinateur personnel en émulation de terminal    *
;*  VT100.                                                                   *
;*                                                                           *
;*  2)                                                                       *
;*  reçoit des caractères à partir d'un clavier                              *
;*                                                                           *
;*  3)                                                                       *
;*  traduit des chiffres hexadécimal en caractère ascii pour les             *
;*  afficher sur le terminal                                                 *
;*                                                                           *
;*  4)                                                                       *
;*  active un compteur binaire sur les DELs branchées sur le port B          *
;*                                                                           *
;*****************************************************************************

; Point d'entrée du programme

            ABSENTRY    Entry           ; point d'entrée pour adressage absolu


            nolist                      ; Désactiver l'insertion de texte dans le
                                        ; fichier .LST
            INCLUDE     'mc9s12c32.inc' ; Inclusion du fichier d'identification des
                                        ; registres
            INCLUDE     'D_BUG12M.MAC'  ; Définition de macros pour des appels simples
                                        ; en assembleur ; getchar, putchar, out2hex
                                        ; out4hex, printf
                                        ; Réactiver l'insertion de texte dans le
            list                        ; fichier .LST

ROMStart    EQU         $4000           ; Adresse absolue pour le début du programme
                                        ; et des constantes


;*****************************************************************************
;*                                                                           *
;*  Déclaration des variables                                                *
;*                                                                           *
;*  data section MY_EXTENDED_RAM à $0800                                     *
;*                                                                           *
;*****************************************************************************
```

```
            ORG     RAMStart

Compt       ds.b    1                       ; Compteur binaire

;*************************************************************************
;*************************************************************************
;*                                                                       *
;*  Début du code dans la section CODE SECTION                           *
;*                                                                       *
;*                                                                       *
;*                                                                       *
;*************************************************************************
;*************************************************************************

            ORG     ROMStart

Entry:
            CLI                 ; permettre les interruptions

            LDSS    #$1000      ; initialisation de la pileau haut
                                ; de ls RAM ($0800-$0FFF)


;*************************************************************************
;
;Init portb en sortie


            LDAB    #$ff        ; 1 = sortie
            STAB    DDRB

;*************************************************************************
;
; Init du SCI (transmetteur et récepteur de caractères sériel)


            CLR     SCIBDH
            LDAB    #$34        ; si bus clk est à 8MHz
            STAB    SCIBDL      ; 9600 BAUDS
            CLR     SCICR1      ; M BIT = 0 POUR 8 BITS
            LDAB    #$0C
            STAB    SCICR2      ; TE , RE

;*************************************************************************
;
; À partir des fonction D_BUG12
;
; 1) Envoyer un message au terminal VT100
; 2) Recevoir des caractères du clavier et en faire l'écho sur le terminal
;    jusqu'à la réception du caractère crlf [ENTER]

Main:
            printf  #Message    ; envoyer un message de bienvenue

ReChar1:                        ; lecture du clavier
            getchar
            CMPB    #$0d        ; si crlf fin de lecture
            BEQ     Mes1
            putchar
            BRA     ReChar1     ; faire un écho
Mes1:       printf  #Message2   ; envoyer message à la prochaine
```

```
;*********************************************************************
;*
;*  Essai des fonctions d'affichage hexadécimal à ASCII


            out2hex Message     ; faire test hex to ascii 1 octets
                                ; mode étendu
            printf  #CRLF        ; Retour à la ligne et nouvelle ligne

            out4hex #$a25f      ; faire test hex to ascii 2 octets
                                ; mode immédiat

;*********************************************************************
;*
;*  Compteur binaire sur le port B


            LDX     #$0000

loop:       INX                  ; petit délai de 65536 comptes
            BNE     loop

            LDAB    Compt        ; Incrémenter la variable Compt
            INCB
            STAB    Compt
            STAB    PORTB        ; Compt vers le port B
            BRA     loop


Message:    dc.b    'Bienvenue au laboratoire de microélectronique',$0A,$0D
            dc.b    'appliquée',$0A,$0D,$00

Message2:   dc.b    $0A,$0D, 'Bravo et à la prochaine !!! ',$0A,$0D,$00

CRLF:       dc.b    $0A,$0D,$00  ; $0A = descendre le curseur d'une ligne
                                 ; $0D = retourner le curseur à la colonne 0
                                 ; $00 = fin de texte pour la fonction printf


            NOLIST
;*********************************************************************
;*                                                                   *
;*  Inclusion du fichier D_BUG12M.ASM                                *
;*                                                                   *
;*********************************************************************

            INCLUDE 'D_BUG12M.ASM' ; Fichier pour la simulation des
                                   ; fonctions D_BUG12
            LIST
;*********************************************************************
;*                                                                   *
;*                 Vecteur d'interruption pour le reset              *
;*                                                                   *
;*********************************************************************

            ORG     $FFFE
            fdb     Entry           ;Reset


             END                    ; fin de compilation
```

1.1.7 Compilation

Une fois que votre programme a été tapé et sauvegardé, vous pouvez le compiler et corriger vos erreurs de syntaxe.

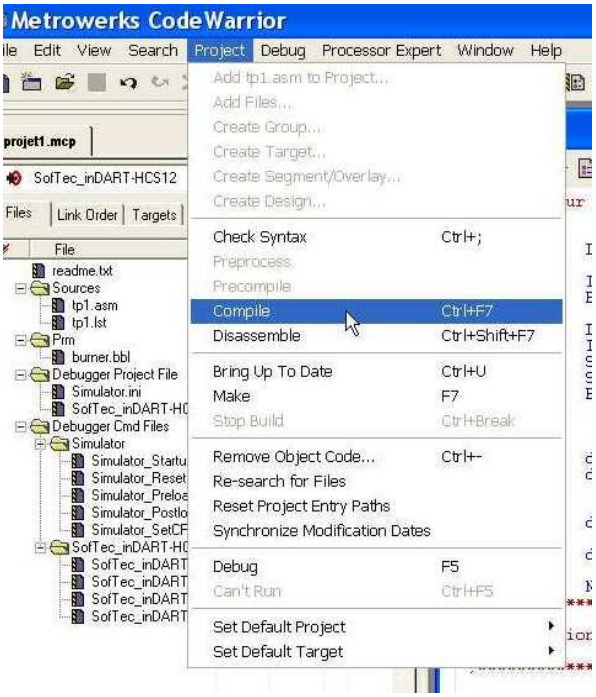


Figure 19. Menu Projet/Compile, /Make et /Debug

- Pour compiler la fenêtre active, cliquer sur **Projet/Compile, Ctrl+F7** ou cliquer sur l'icône **Compile**

Figure 20. Compile

- Pour tout recompiler vos programmes sources, cliquer sur **Projet/Make, F7** ou cliquer sur l'icône **Make**

Figure 21. Make

Avant de lancer le déverminer (Debug), il faut indiquer la connexion vers laquelle le déverminer téléchargera et déverminera votre programme dans votre environnement. Vous avez préalablement installé deux types de connexions, une connexion en mode simulation du microcontrôleur (Simulator), et une connexion via le bus USB en mode réel de téléchargement et de déverminage du microcontrôleur (SofTec).

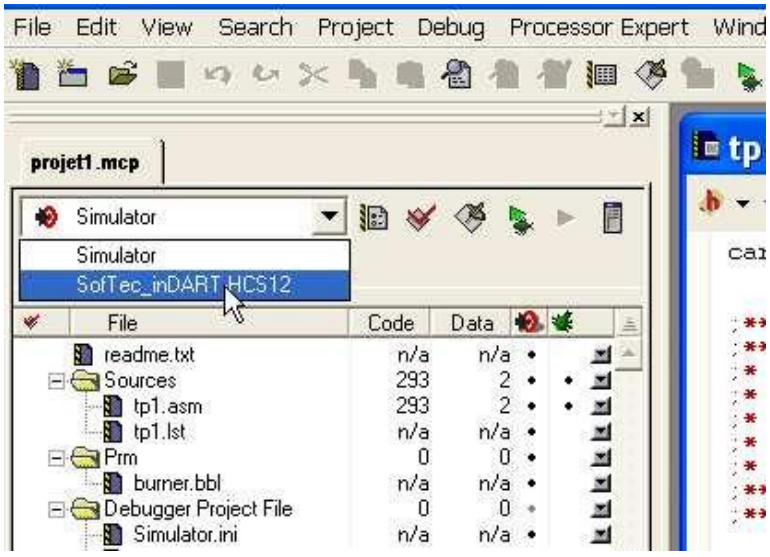


Figure 22. Choix de la connexion pour le déverminer

1.1.8 Téléchargement

1.1.8.1 Connexion virtuelle (Simulator)

Pour la connexion vers le simulateur de microcontrôleur : 1) assurez-vous d'avoir sélectionné la connexion (Simulateur) dans votre environnement, 2) branchez à cette connexion un terminal et des entrées/sorties virtuelles grâce aux commandes suivantes dans le fichier **Full_Chip_Simulator_SetCPU.cmd** :

```
// At startup the commands written below will be executed
OPEN Terminal 40 40 40 40
OPEN IO_LED
PORT 1
DDR 3
```

1.1.8.2 Connexion réelle (SofTec)

Pour la connexion vers le microcontrôleur : 1) assurez-vous d'avoir sélectionné la connexion (SofTec) dans votre environnement, 2) reliez le câble USB de votre PC à la carte HCS12 STARTER KIT.

1.1.8.3 Téléchargement vers la connexion choisie

Le téléchargement de votre programme s'effectuera en invoquant le dévermineur. Pour invoquer le dévermineur, sélectionner **Projet/Debug, F5** ou cliquer sur l'icône **Debug.**

Figure 23. Debug

À la compilation, une fenêtre de construction de projet s'ouvre et monitorise la compilation de votre programme. Si des erreurs de syntaxe ou des mises en gardes sont décelées, le ou les numéros de ligne ainsi que le ou les types d'erreurs vous sont présentés.

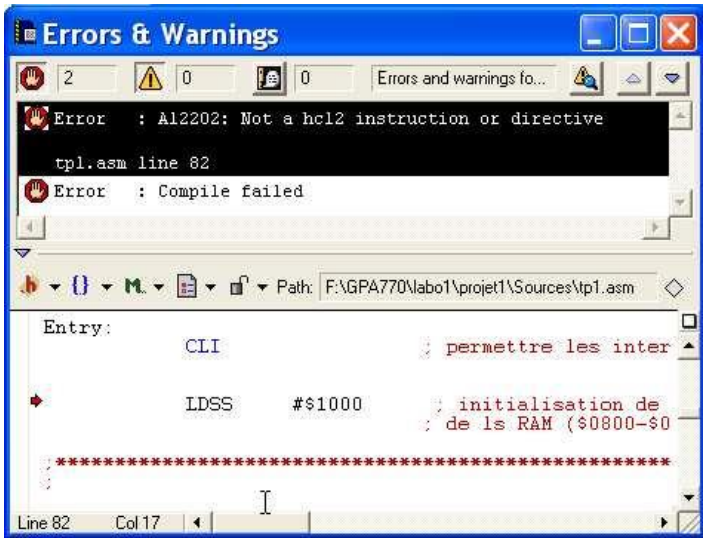


Figure 24. Erreurs et mises en gardes

CodeWarrior repère ces lignes et pointe les erreurs à l'aide d'un curseur rouge. Vous pouvez corriger vos erreurs dans cette fenêtre ou bien dans votre fenêtre d'édition. **Corrigez le LDSS par LDS et recompiler.**

À ce niveau, le fichier de téléchargement S19 ainsi que le fichier de listing LST ont été créés dans vos répertoires.

On retrouve dans le fichier LST cinq colonnes :

- Première colonne (**Abs**.) : Numéros de lignes absolues incluant les INCLUDEs et les MACROs de votre programme.
- Deuxième colonne (**Rel.**) : Numéros de lignes relatives de votre programme source.
- Troisième colonne (**Loc.**) : Adresses absolues en mémoire de votre code objet assembleur.
- Quatrième colonnes (**Obj. Code**) : Codes objets en assembleur d'une ligne source.
- Cinquième colonnes (**Sources line**) : Code source que vous avez tapé.

Fichier Lab1.lst

```
Freescale HC12-Assembler
(c) Copyright Freescale 1987-2005

 Abs. Rel.   Loc    Obj. code   Source line
 ---- ----   ------ ---------   -----------
    1    1                      ;*************************************************************************
    2    2                      ;*                                                                       *
    3    3                      ;*  Auteur : Maurice Tremblay                                            *
    4    4                      ;*  Date :   avril 2007                                                  *
    5    5                      ;*                                                                       *
    6    6                      ;*                                                                       *
    7    7                      ;*  Ce programme :                                                       *
    8    8                      ;*                                                                       *
    9    9                      ;*  1)                                                                   *
   10   10                      ;*  affiche un message de bienvenue par une transmission                 *
   11   11                      ;*  sérielle de caractères du microcontrôleur, port SCI, vers le port de *
   12   12                      ;*  communication COM1 d'un ordinateur personnel en émulation de terminal*
   13   13                      ;*  VT100.                                                               *
   14   14                      ;*                                                                       *
   15   15                      ;*  2)                                                                   *
   16   16                      ;*  reçoit des caractères à partir d'un clavier                          *
   17   17                      ;*                                                                       *
   18   18                      ;*  3)                                                                   *
   19   19                      ;*  traduit des chiffres hexadécimal en caractère ascii pour les         *
   20   20                      ;*  afficher sur le terminal                                             *
   21   21                      ;*                                                                       *
   22   22                      ;*  4)                                                                   *
   23   23                      ;*  active un compteur binaire sur les DELs branchées sur le port B      *
   24   24                      ;*                                                                       *
   25   25                      ;*************************************************************************
   26   26
   27   27                      ; Point d'entrée du programme
   28   28
   29   29                                  ABSENTRY    Entry         ; point d'entrée pour adressage absolu
         .
         .
         .
         .
         .
         .
 5538  118                       ReChar1:                          ; lecture du clavier
 5539  119                                  getchar
 5540   15m a00401E 15FB BDE0  +            jsr     [jgetchar,pcr]  ; retourne le caractère dans le registre b
 5541  120  a004022 C10D                   CMPB    #$0d        ; si crlf fin de lecture
 5542  121  a004024 2706                   BEQ     Mes1
 5543  122                                  putchar
 5544   19m                     +                                              ; chargé dans b
 5545   20m a004026 15FB BDDA  +            jsr     [jputchar,pcr]  ; le caractère transmis est retourné dans b
 5546  123  a00402A 20F2                   BRA     ReChar1     ; faire un écho
         .
         .
         .
         .
         .
         .
         .
 5586  155  a004059 4269 656E   Message:    dc.b    'Bienvenue au laboratoire de microélectronique',$0A,$0D
              00405D 7665 6E75
              004061 6520 6175
              004065 206C 6162
              004069 6F72 6174
              00406D 6F69 7265
              004071 2064 6520
              004075 6D69 6372
              004079 6FE9 6C65
              00407D 6374 726F
              004081 6E69 7175
              004085 650A 0D
 5587  156  a004088 6170 706C              dc.b    'appliquée',$0A,$0D,$00
              00408C 6971 75E9

         .
         .
         .
         .                       END
```

1.1.8.4 Déverminage

À l'amorce, le dévermineur de FreeScale ouvre ses multiples fenêtres et exécute les fichiers de commandes .CMD vus plus tôt, de votre projet. Vous pouvez voir dans la fenêtre d'état du bas (Command) se dérouler l'exécution des commandes incluses dans les fichiers .CMD.

Ces commandes ont les fonctions suivantes :

1) conditionnent et téléchargent le fichier.S19 au microcontrôleur dans sa mémoire FLASH,
2) chargent le fichier lab1.asm dans la fenêtre (**Source**),
3) chargent une version compilée de lab1.asm dans la fenêtre (**Assembly**),
4) monitorisent les variables dans la fenêtre (**Data**),
5) monitorisent la mémoire à partie de l'adresse $0080 dans la fenêtre (**Memory**),
6) préconditionnent le compteur de programme à l'adresse de départ $4000 pour le pointeur d'instruction IP (**I**nstruction **P**ointer) et le compteur de programme PC (**P**rogram **C**ounter).

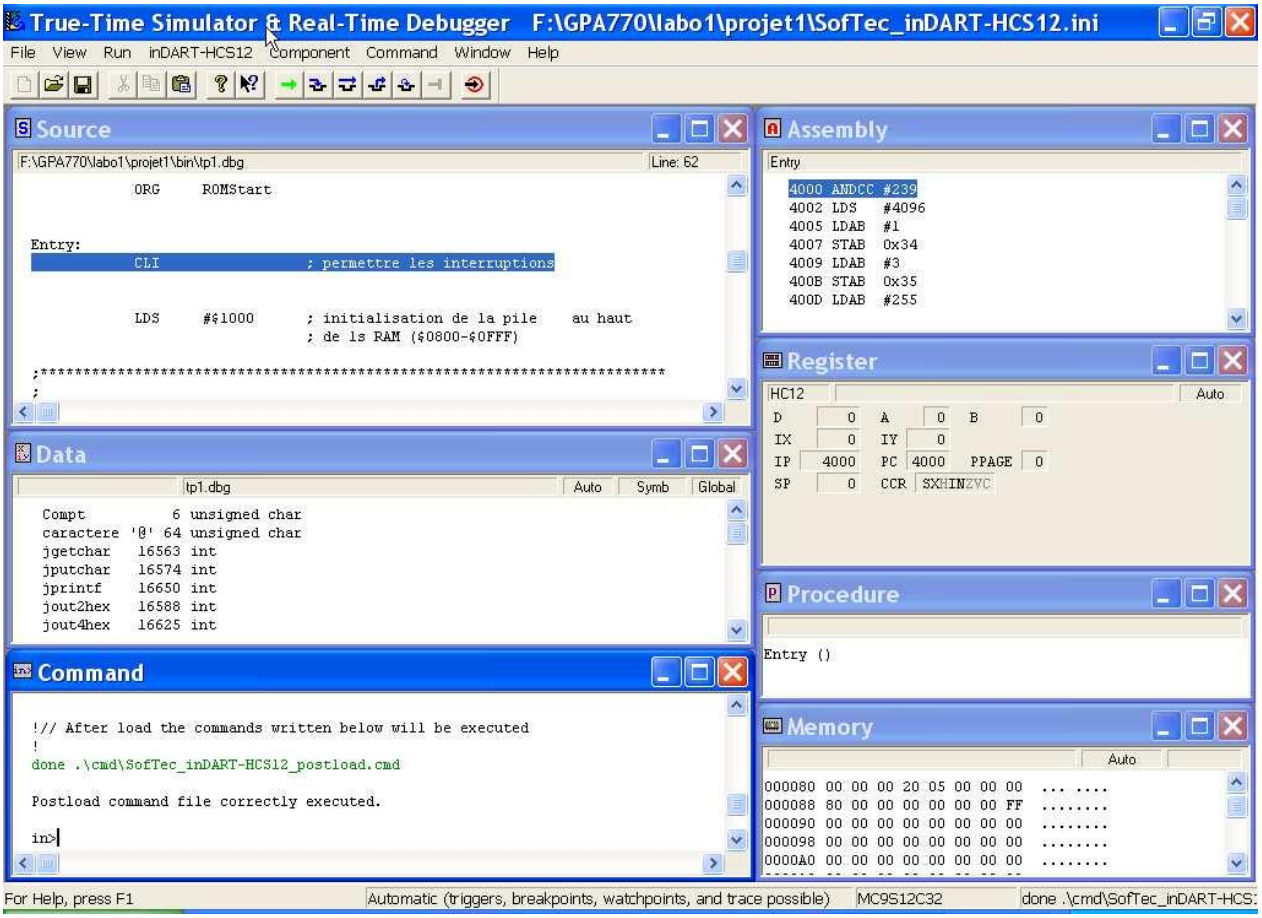


Figure 25. Dévermineur

La fenêtre (**Register**) affiche le contenu des registres internes au CPU12. Ces registres ne font pas partie de la mémoire du microcontrôleur. Ils appartiennent à l'ordinateur CPU12 du microcontrôleur. Ils ne possèdent pas d'adresses mais des noms, A, B, D …. Etc.

Le point d'entrée de votre programme est **ip 4000**, c'est à dire, charge $4000 dans le pointeur d'instruction. Vous pouvez vérifier que ip = 4000 dans la fenêtre (**Register**).

1.1.8.5 Déplacement dans le programme

Au départ, le devermineur ne démarre pas votre programme. Son pointeur d'instruction est prêt à exécuter l'instruction contenue dans l'adresse $4000 soit **CLI** qui stipule de permettre toutes les interruptions.

Pour exécuter cette instruction seulement, vous pouvez taper dans la fenêtre de Commande, la commande **STEPINTO** ou cliquer une fois sur la flèche bleue à la gauche de la barre d'outils (**Single Step**).

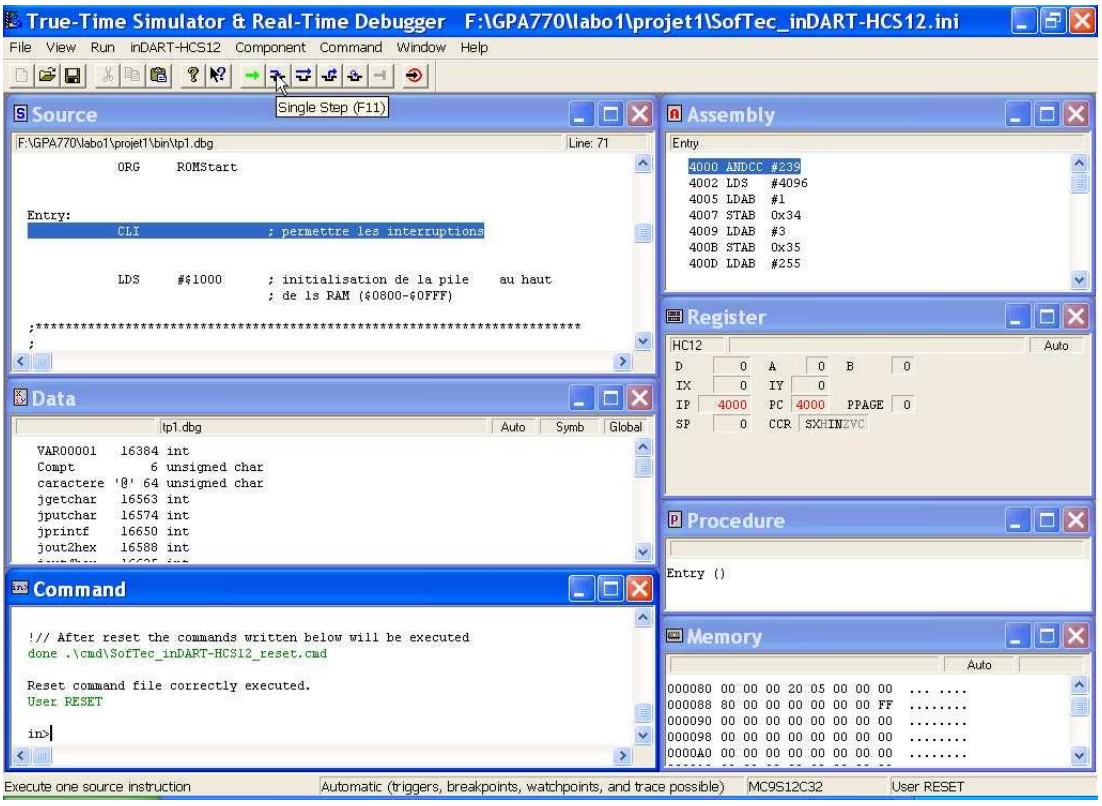


Figure 26. Interaction du dévermineur

Après l'exécution de cette instruction, on remarque que les registres CCR et IP de la fenêtre (Register) sont passés de SXH**IN**ZVC à SXHI**N**ZVC et de $4000 à $4002 reflétant ainsi le

nouvel état du microcontrôleur qui a mis à zéro le bit I du CCR et qui est prêt à exécuter la prochaine instruction à l'adresse $4002. Vous venez d'expérimenter une des trois principales façons de se déplacer dans un programme, le pas à pas.

Une deuxième façon de se déplacer, est d'exécuter le code tant que l'on ne rencontre pas une instruction donnée. Naviguez dans une des fenêtres de code et cliquez sur une instruction avec le bouton gauche de la souris. Cliquer une deuxième fois avec le curseur de droite et sélectionner **Run To Cursor**. Le programme s'exécutera de sa position d'arrêt (ligne en bleu) à la position de votre curseur pour se remettre en position d'attente.

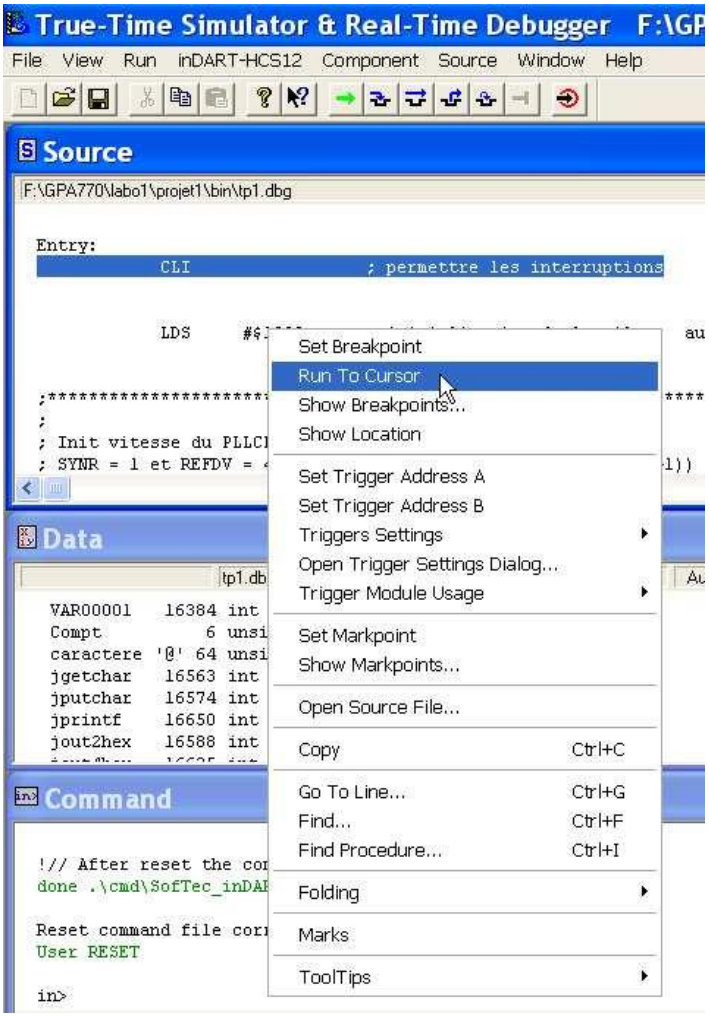


Figure 27. Exécuter le code jusqu'à cette ligne

Finalement, une troisième façon d'avancer dans votre programme est d'insérer des points d'arrêts dans le programme à des adresses particulières et de lancer votre programme avec la commande GO dans la fenêtre de (**Command**) ou avec la flèche verte à la gauche dans la barre d'outils.

L'insertion de points d'arrêts peut se faire de deux façons, une avec le mode commande et une autre avec la souris. Premièrement, avec le mode commande, on tape dans la fenêtre de commande, **BS** suivi de l'adresse à rejoindre :

Ex. : BS $xxxx où xxxx est l'adresse de ReChar1

ou de son étiquette sachant qu'une étiquette est synonyme d'une adresse.

Ex. : BS ReChar1

Deuxièmement, avec la souris, naviguer dans une des fenêtres de code et cliquez sur une instruction avec le bouton droit de la souris et sélectionner **Set Break Point**. Un signet rouge apparaîtra devant l'instruction. Après un GO, le programme s'exécutera de sa position d'arrêt (ligne en bleu) jusqu'au premier point d'arrêt rencontré.

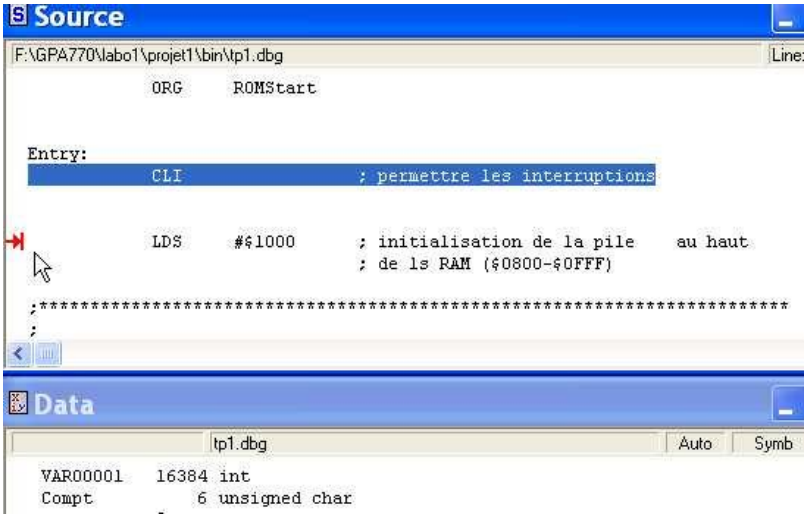


Figure 28. Insertion d'un point d'arrêt

Enfin, pour enlever un des points d'arrêt, on tape la commande **BC** dans la fenêtre de commande.

Ex. : BC $xxxx où xxxx est l'adresse de ReChar1

Ou avec le bouton droit de la souris sur le signet en sélectionnant **Delete Breakpoint** .

L'interaction bidirectionnelle des fenêtres du dévermineur monitorise et conditionne l'état de votre programme.

On peut changer la valeur d'un registre de contrôle du CPU12 en double cliquant sur l'adresse d'un registre.

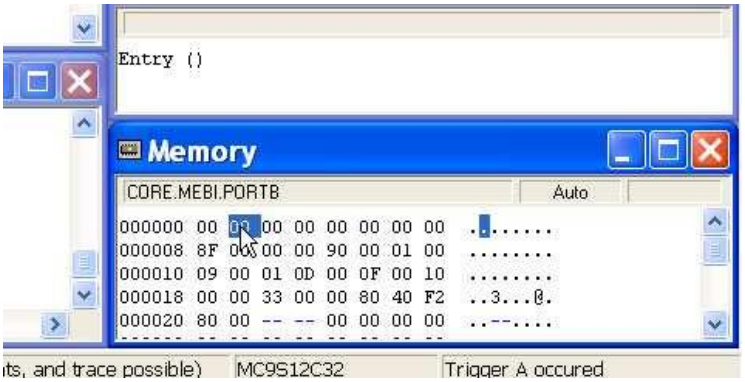


Figure 29. Changer la valeur du registre d'entrée/sortie PORTB

ou bien en changeant la valeur d'une allocation mémoire en double-cliquant sur celle-ci dans la fenêtre (**Memory**) .

Noter que le dévermineur est l'outil le plus important à maîtriser. Le temps que vous passerez à apprendre à le contrôler, vous sera rendu au centuple dans les prochaines séances de laboratoire. Consultez le manuel **Manual Engine HC12.pdf** pour de plus amples informations.

Explorez-le à fond !

1.2 LABORATOIRE #1b Programme simple avec branchements de base

Jusqu'à présent, vous avez mis en place votre environnement, ouvert un projet et testé cet environnement. Vous êtes fin prêt à programmer en langage assembleur.

Ce laboratoire est conçu pour faciliter l'utilisation d'instructions pour les branchements dans les situations incluant les cas « if-then-else ».

Avec quelques modifications, vous pourrez utiliser le programme résultant en tant que sous-routine dans le laboratoire final pour créer le système de navigation d'un robot mobile dans un labyrinthe. La sous-routine pourra être utilisée pour lire les valeurs des détecteurs infrarouges (IR) afin de prendre des décisions appropriées pour la navigation d'un robot.

1.2.1 Simulation de la lecture de capteurs infrarouges.

Débutez votre programme à l'adresse $4000 avec la directive de compilation **ORG**.
À la fin du laboratoire, votre programme doit accomplir les tâches suivantes :

1- Lire une série de trois nombres hexadécimaux situés dans les adresses mémoire $0091, $0093 et $0095 et les mettre respectivement en mémoire aux adresses $0800, $0801, et $0802. Vous pouvez considérer que les premières adresses mémoire correspondent aux valeurs du voltage de sorties des trois détecteurs Irs, et que les trois dernières sont des registres pour conserver les valeurs des détecteurs.

2- Lire les trois valeurs des nouvelles adresses mémoire et vérifier chacune par rapport à une valeur seuil de $55, laquelle doit être définie en utilisant la directive de compilation **EQU** au début du programme. La sortie des capteurs Irs est une valeur hexadécimale non signée et indique la proximité des obstacles environnants.

3- Si la valeur à l'adresse mémoire $0800 est plus grande que la valeur seuil, calculez la somme des cinq nombres suivant (1 + 2 + 3 + 4 + 5) et déposer le résultat à l'adresse $0800.

4- Si la valeur à l'adresse mémoire $0801 est plus petite que la valeur seuil, remplacer la par $00.

5- Si la valeur à l'adresse mémoire $0802 est plus grande que la valeur seuil, soustraire $10 de cette valeur et déposer le résultat de la soustraction à cette même adresse mémoire.

1.3 LABORATOIRE #1c Sous-routines et passage de paramètres

Ce laboratoire vous fournit l'occasion d'utiliser des sous-routines avec les techniques de passage des paramètres par valeurs et par références.

Le programme résultant pourra (avec quelques modifications) être utilisé dans le laboratoire final pour contrôler le couple des moteurs DC du robot.

1.3.1 Passage de paramètres par valeurs et par références

La technique de passage de paramètres par valeur utilise les registres A, B, X, et Y du CPU pour passer une **copie** des valeurs originales aux sous-routines appelées.

La technique de passage de paramètres par références fournit aux sous-routines appelées les adresses de l'endroit où résident les valeurs originales dans la mémoire ou bien l'adresse de la première valeur originale dans le cas d'un tableau de valeurs.

Lorsque le programme appelant doit conserver certaines informations avant de passer le contrôle à la sous-routine appelée, il peut soit conserver ces valeurs sur la pile ou soit les placer dans des adresses mémoire.

Lorsque que la sous-routine appelée doit modifier des registres du CPU et que ces registres ne sont pas retournés à l'appelant, elle peut les sauvegarder sur la pile au début de la sous-routine et les restituer avant le RTS.

1.3.2 Simulation du contrôle de deux moteurs DC

Afin de simuler un contrôle de moteur, nous assumons que stoker les valeurs de deux octets (16 bits) dans les adresses mémoire M:M+1 à M+58:M+59, affectera les vitesses du moteur gauche du robot. De façon similaire, stocker les valeurs de deux octets (16 bits) dans les adresses mémoire M+60:M+61 à M+118:M+119, contrôlera les vitesses du moteur droit du robot. Il est à noter que l'adresse la plus basse contient l'octet le plus significatif de la vitesse du moteur.

M : M+1	
$17	**$70**

Figure 30. Exemple de la première vitesse du moteur gauche en mémoire

La vitesse d'un moteur devra suivre un profile trapézoïdal pour simuler un déplacement doux du robot. Accélération douce (PA) suivie d'une vitesse constante (PC) et d'une décélération douce (PD).

Considérez les chiffres suivants pour contrôler la vitesse d'un des moteurs.

Une vitesse maximum avant = 8000 déc.
Une vitesse nulle = 6000 déc.
Une vitesse maximum arrière = 4000 déc.

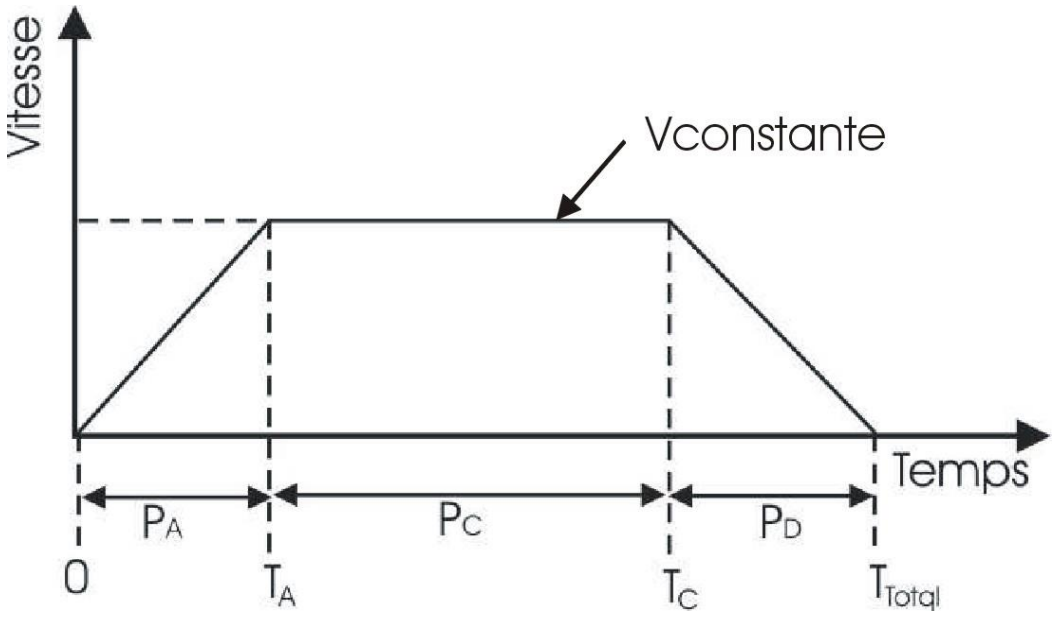


Figure 31. Exemple du profil de la vitesse d'un moteur

À partir des deux paramètres suivants, la vitesse constante 16 bits non signée (**Vconstante** en RAM) et le temps total 16 bits non signé (**Ttotal** en RAM) :

1- Faire une première sous routine à l'aide de la technique de passage des paramètres par valeurs qui sera en mesure de calculer les temps PA, PC et PD (c. à d. trouver **TA** et **TC**) sachant que les temps d'accélération et de décélération représentent chacun 20% du temps total. Sauvegarder en RAM les valeurs 16 bits des temps **TA**, **TC** et Ttotal.

   La sous-routine devra aussi calculer l'intervalle de vitesse **DelataV** nécessaire pour réajuster l'accélération et la décélération 10 fois chacune et sauvegarder cette valeur de 16 bits en RAM.

2- Faire une seconde sous-routine à l'aide de la technique de passage de paramètres par références (adresse **DeltaV** et adresse d'un tableau de vitesse **VMG** en RAM) qui sera en mesure de calculer les 10 vitesses d'accélération, les 10 vitesses de la vitesse constante et les 10 vitesses de la décélération. Sauvegarder les 30 vitesses pour le moteur droit à partir des adresses VMD et les 30 vitesses pour le moteur gauche à la suite du tableau **VMD** c. à d. à **VMG**.

Pour le moment, ignorez les valeurs de temps qui correspondent aux vitesses des moteurs. Dans un premier système de contrôle des moteurs, le programme écrira constamment les valeurs de la vitesse aux adresses (M:M+1) du moteur gauche et (M+60:M+61) du moteur droit.

1.4 LABORATOIRE #1d Sous-routines d'entrées/sorties D-BUG12M

Ce laboratoire vous fournit l'occasion d'utiliser quelques sous-routines d'entrées/sorties (simulées) D_BUG12(M) de Motorola qui pourront vous aider tout au long de la session dans l'élaboration de vos programmes, soit **getchar, putchar, out2hex, out4hex et printf**.

Ces routines permettent l'utilisation simple d'un terminal intelligent bidirectionnel pour communiquer avec votre microcontrôleur HCS12.

1- Modifier le programme précédent pour demander à un utilisateur un choix parmi trois vitesses et trois temps total comme suit :

```
Choisissez une vitesse constante :
Rapide (1), Moyenne (2), lente (3)
2
Choisissez un temps total :
Rapide (1), Moyenne (2), lente (3)
2
```

Prenez les chiffres suivants comme constantes dans votre programme **<u>à la fois pour les vitesses et pour les temps.</u>**

RAPIDE	$1F40	(8,000 décimal)
MOYENNE	$1B58	(7,000 décimal)
LENTE	$1964	(6,500 décimal)

2- En réponse aux choix de la vitesse et du temps total, votre programme devra appeler une sous-routine avec passage de paramètres par références pour afficher le profile de la vitesse des deux moteurs. De plus, vous devrez afficher le temps d'accélération TA, le temps de vitesse constante TC, le temps de vitesse total Ttotal et finalement le delta de voltage pour l'accélération et la décélération DeltaV comme suit en supposant que le choix était de 2 pour la vitesse et de 2 pour le temps total :

```
Profil de la vitesse

            Droit  gauche

Vitesse 01 :  170C :  17D4
Vitesse 02 :  16A8 :  1838
Vitesse 03 :  1644 :  189C
Vitesse 04 :  15E0 :  1900
Vitesse 05 :  157C :  1964
Vitesse 06 :  1518 :  19C8
Vitesse 07 :  14B4 :  1A2C
Vitesse 08 :  1450 :  1A90
```

```
Vitesse 09 :  13EC :  1AF4
Vitesse 0A :  1388 :  1B58
Vitesse 0B :  1388 :  1B58
Vitesse 0C :  1388 :  1B58
Vitesse 0D :  1388 :  1B58
Vitesse 0E :  1388 :  1B58
Vitesse 0F :  1388 :  1B58
Vitesse 10 :  1388 :  1B58
Vitesse 11 :  1388 :  1B58
Vitesse 12 :  1388 :  1B58
Vitesse 13 :  1388 :  1B58
Vitesse 14 :  1388 :  1B58
Vitesse 15 :  13EC :  1AF4
Vitesse 16 :  1450 :  1A90
Vitesse 17 :  14B4 :  1A2C
Vitesse 18 :  1518 :  19C8
Vitesse 19 :  157C :  1964
Vitesse 1A :  15E0 :  1900
Vitesse 1B :  1644 :  189C
Vitesse 1C :  16A8 :  1838
Vitesse 1D :  170C :  17D4
Vitesse 1E :  1770 :  1770

TA :  0578
TC :  15E0
Ttotal :  1B58
DeltaV :  0064
```

1.5 LABORATOIRE #1e Contrôleur à logique flou avec les instructions du HCS12

Ce laboratoire est conçu pour vous permettre l'étude et la mise en application d'un contrôleur à logique flou. À cette fin, vous utiliserez le jeu d'instructions spécialisées et intégrées au 68HCS12.

Ce laboratoire n'exécutera qu'une seule itération de contrôle et sera utilisé comme une partie du programme final pour le contrôle du robot.

1.5.1 Contrôle de direction d'un robot

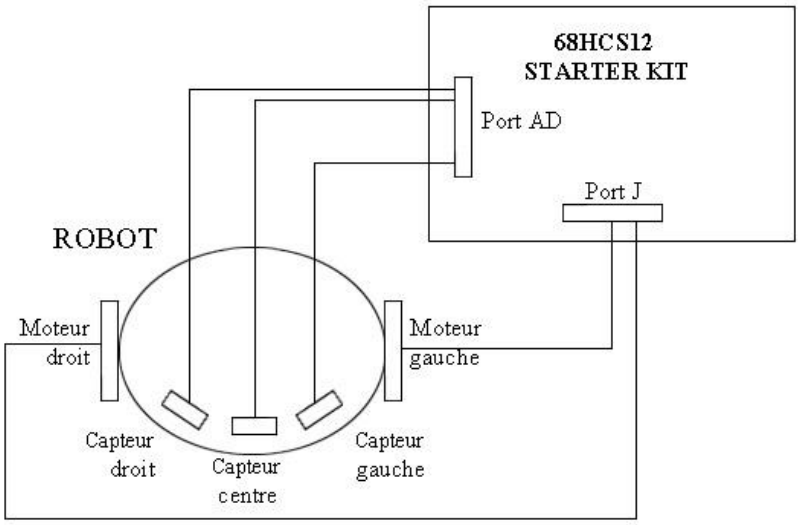


Figure 32. Localisation des capteurs IRs du robot[6]

Le robot est équipé de trois capteurs infrarouges (IR)s, Sharp GP2Y0A21YK, servant à détecter la proximité d'obstacles.

Chacun des capteurs possède une lumière IR directionnelle et un capteur focal de lumière IR réfléchie associé, pulsé à 40ms. La sortie en voltage des capteurs est branchée à trois différents canaux du convertisseur analogique à numérique (A/D) du port PORTAD du 68HCS12. Plus la

[6] PACK Daniel J. et BARRETT Steven F. , **68HC12 MICROCONTROLLER – THEORY AND APPLICATIONS** , Prentice Hall, 2002, pages 158 à 163.

grandeur focal captée par les convertisseurs A/Ds est au foyer plus la proximité d'un obstacle est forte et vice-versa.

Capteur infrarouge Sharp GP2Y0A21YK
Distance de détection 10 à 80 cm

Distance(cm)	Voltage(mV)
Infinie	250
90	360
80	380
70	420
60	500
50	600
40	880
30	900
20	1400
10	2000

Tableau 1 Distances des obstacles vs voltages lus par les capteurs

Chaque convertisseur possède une mémoire RAM de 8 bits pour stocker une valeur de voltage non signée comprise entre $00 et $FF. Nous établirons plus tard une corrélation entre les voltages du tableau précédant et les valeurs de voltages de $00 à $FF utilisées par le contrôleur flou. Pour l'instant nous simulerons ces valeurs de voltages en mémoire RAM.

1.5.1.1 Nombres de variables requises et attribution des noms

Nous avons trois variables pour les voltages d'entrées de chaque capteur : **Vcapt_droit, Vcapt_centre, Vcapt_gauche**.

En utilisant les trois fonctions d'appartenance d'entrée du graphique suivant pour chacun des capteurs IRs, nous obtenons neuf variables pour les degrés d'appartenances **: D_Loin, D_Midi, D_Pres, C_Loin, C_Midi, C_Pres, G_Loin, G_Midi, G_Pres**.

Notez que la non-linéarité entre les fonctions **LOIN**, **MIDI**, et **PRES** respecte la non-linéarité des voltages de sorties des capteurs IRs du tableau précédent.

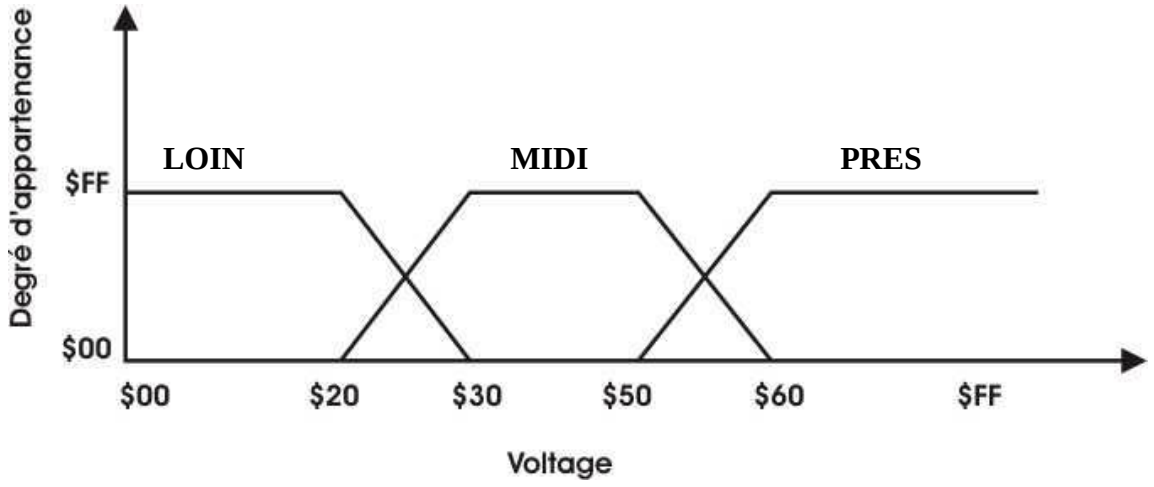


Figure 33. Définition des fonctions d'appartenance d'entrées pour chacune des variables du contrôleur flou.

En utilisant les trois singletons pour les poids de la moyenne pondérée suivants, on défini trois constantes **: AGAUCHE = $10, DEVANT=$80, ADROITE=$F0.**

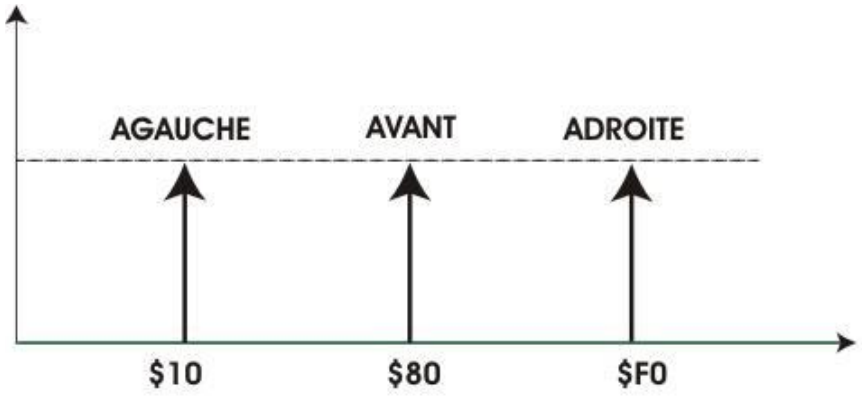


Figure 34. Singletons pour défuzzifier la sortie du contrôleur flou.

1.5.1.2 Développement des règles du contrôleur flou

Comme nous avons trois jeux de fonctions d'appartenance en entrée, il faut définir 3 X 3 X 3 règles pour le contrôleur flou, soit 27 règles en tout.

Tableau des Règles du contrôleur floues				
Règles	**Capteur gauche**	**Capteur central**	**Capteur droit**	**Déplacement du robot**
Règle 1	LOIN	LOIN	LOIN	SF_AVANT
Règle 2	LOIN	LOIN	MIDI	SF_AGAUCHE
Règle 3	LOIN	LOIN	PRES	SF_AGAUCHE
Règle 4	LOIN	MIDI	LOIN	SF_ADROITE
Règle 5	LOIN	MIDI	MIDI	SF_AGAUCHE
Règle 6	LOIN	MIDI	PRES	SF_AGAUCHE
Règle 7	LOIN	PRES	LOIN	SF_ADROITE
Règle 8	LOIN	PRES	MIDI	SF_AGAUCHE
Règle 9	LOIN	PRES	PRES	SF_AGAUCHE
Règle 10	MIDI	LOIN	LOIN	SF_ADROITE
Règle 11	MIDI	LOIN	MIDI	SF_ADROITE
Règle 12	MIDI	LOIN	PRES	SF_AGAUCHE
Règle 13	MIDI	MIDI	LOIN	SF_ADROITE
Règle 14	MIDI	MIDI	MIDI	SF_ADROITE
Règle 15	MIDI	MIDI	PRES	SF_AGAUCHE
Règle 16	MIDI	PRES	LOIN	SF_ADROITE
Règle 17	MIDI	PRES	MIDI	SF_ADROITE
Règle 18	MIDI	PRES	PRES	SF_AGAUCHE
Règle 19	PRES	LOIN	LOIN	SF_ADROITE
Règle 20	PRES	LOIN	MIDI	SF_ADROITE
Règle 21	PRES	LOIN	PRES	SF_ADROITE
Règle 22	PRES	MIDI	LOIN	SF_ADROITE
Règle 23	PRES	MIDI	MIDI	SF_ADROITE
Règle 24	PRES	MIDI	PRES	SF_ADROITE
Règle 25	PRES	PRES	LOIN	SF_ADROITE
Règle 26	PRES	PRES	MIDI	SF_ADROITE
Règle 27	PRES	PRES	PRES	SF_ADROITE

Tableau 2 Règles du contrôleur flou

1.5.1.3 Affichage des résultats du contrôleur flou

Une fois le programme du contrôle flou de navigation du robot en place, avec les fonctions de D-BUG12M, afficher à l'écran le texte suivant :

```
Entrées = Vcap_gauche : XX    Vcap_centre : XX    Vcap_droite : XX
Sortie = XX
```

Les XXs en entrée représentent les valeurs de voltages hexadécimales des capteurs. Les XXs de la sortie représentent la direction que le robot doit prendre de $10 à $F0 où : $10 = braquage à gauche maximum, $F0 = braquage à droite maximum.

1.6 Rapport pour le laboratoire #1

1.6.1 Travail à remettre

Pour ce premier laboratoire, vous devez remettre :

1- Un imprimé fonctionnel et très bien commenté de vos programmes LAB1b.ASM, LAB1d.ASM et LAB1e.ASM ainsi qu'un imprimé de votre rapport.

N.B. Pour l'imprimé, prenez soin d'utiliser une police à pas fixe (COURRIER) capable de reproduire la plus grande ligne de texte de votre programme sur une seule ligne de l'imprimé. L'impression en mode paysage est acceptée.

2- Une **copie sur MOODLE** de vos programmes LAB1b.ASM, LAB1d.ASM et LAB1e.ASM. ainsi que de votre rapport *.docx.

1.6.2 Barème de correction

Programmes (a/b)

Fonctionnalité, incluant une présentation orale au chargé de TPs……..15%
Code structuré et commentaires……..5%
Rapport…….8%
Introduction……...1%
Objectifs……1%
Méthodes……...3%
Discussion…….2%
Conclusion……1%

Programmes (c/d)

Fonctionnalité, incluant une présentation orale au chargé de TPs……..15%
Code structuré et commentaires……..5%
Rapport…….8%
Introduction……...1%
Objectifs……1%
Méthodes……...3%
Discussion…….2%
Conclusion……1%

Programme (e)

Fonctionnalité, incluant une présentation orale au chargé de TPs……..15%
Code structuré et commentaires……..5%
Rapport…….8%
Introduction……...1%
Objectifs……1%
Méthodes……...3%
Discussion…….2%
Conclusion……1%

Présentation globale des rapports…….10%
Présentation globale du code.……6%

Fin du laboratoire #1

---

2 LABORATOIRE #2 Exploitation en temps réel du matériel et des ports d'entrées/sorties

Le but de ce laboratoire est d'exploiter des composantes de base, en mode interrogation (par le balayage de drapeaux) ou en mode temps réel (par des interruptions). Vous utiliserez les ports d'entrées/sorties et le TIM (Timer Interrupt Module).

Le premier permettra l'utilisation des commutateurs disponibles sur la carte alors que le dernier servira à la génération d'un PWM (Pulse Width Modulator) pour le contrôle de vitesse des moteurs du robot.

2.1 LABORATOIRE #2a Interrogation d'un bouton poussoir.

Ce laboratoire est conçu pour vous permettre l'étude et la mise en application d'un port d'entrées/sorties avec la technique de mode interrogation (pooling). Cette technique consiste à interroger de façon itérative l'état d'un périphérique le plus souvent possible allant jusqu'à l'interroger continuellement au travers une boucle d'interrogation.

1- Attendre au début du programme qu'un bouton poussoir soit poussé avec la technique d'interrogation (polling).
2- Une fois le bouton poussé, votre programme devra écrire à l'écran un message d'accuser réception.

2.1.1 Mode réel

Vous devrez utiliser le commutateur PP0 présent sur la carte d'évaluation, qui correspond au bit PTP0 du port P. Ce commutateur et déjà branché physiquement au port P comme le montre la figure 36 et est normalement ouvert:

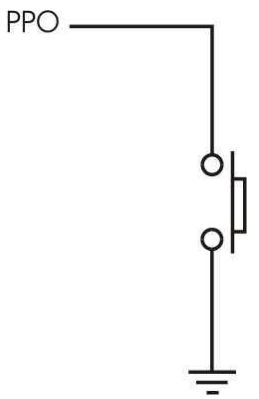


Figure 35. Bouton poussoir de la carte d'évaluation

Notez que ce bouton utilise le mode pull up interne du microcontrôleur 68HCS12.

Initialisez le port P comme suit:

1) Mode pull-ups pour la broche 0 du port P
2) Port P broche 0 en entrée

Vous devez donc lire le port P (registre PTP bit 0) tant et aussi longtemps que l'entrée ne passe pas de 1 à 0 avant de poursuivre votre programme.

2.1.2 Mode simulation

En mode simulation on doit brancher des boutons poussoirs au port P. Insérez les lignes de commandes suivantes dans le fichier **Simulator_SetCPU.cmd** pour brancher les boutons poussoirs au simulateur.

```
OPEN Push_Buttons        Module des boutons poussoirs
PBPORT $0258             Adresse du port P (PTP)
```

Figure 36. Boutons poussoirs en mode simulation

N.B. Les bouton poussoirs du simulateur sont normalement fermés. Il faut donc attendre une transition de 0 à 1 avant de poursuivre le code à l'inverse des boutons poussoirs de la carte de développement.

2.1.3 Mode simulation et réel

Comme vous avez deux type de boutons poussoirs différent, un pour le mode simulation qui donne une transition de 0 à 1 et un pour le mode réel qui donne une transition 1 à 0, il vous faudra deux bouts de code différents pour tester ces boutons.

Il existe des directives de compilation pour vous aider à compiler les lignes de votre code soit pour les boutons poussoirs du simulateur ou pour les boutons poussoirs de la carte d'évaluation dans le mode réel.

Procédez comme suit :

1- Au début du programme déclarez une étiquette ‘Simulateur’. Si l’étiquette égale 1 on compilera les lignes de code spécialisés pour le simulateur, sinon on compilera les lignes de code spécialisés pour la carte d’évaluation.

Ex. : Simulateur EQU 1 ; si Simulateur = 1 on compile le code du
; simulatur

2- Dans votre code utilisez les directives de compilation suivantes, if, else et endif :

Exemple :

```
if Simulateur = 0
                        ; on compile ce bout de code
....                    ; attendre PP0 SofTec_inDART-HCS12
                        ; transition de 1 à 0 pour PTP0
else
                        ; sinon, on compile ce bout de code
....                    ; attendre PP0 Simulator
                        ; transition de 0 à 1 pour PTP0
endif                   ; fin du IF, ELSE

printf ....             ; message d'accusé réception de PTP0
```

N.B. **if**, **else** et **endif** ne sont pas des lignes de code mais des directives de compilation.

2.2 LABORATOIRE #2b E/S, afficheur LCD et interruptions externes

Le but de ce laboratoire est de continuer d'expérimenter les possibilités des ports d'entrées/sorties en plus des différentes caractéristiques qu'offrent les interruptions du microcontrôleur HCS12.

À partir d'une interruption externe, votre programme devra écrire un message approprié sur un afficheur à cristaux liquide LCD.

Plus tard, cette interruption servira comme bouton d'arrêt d'urgence pour le robot mobile et l'afficheur LCD servira à communiquer des informations à l'utilisateur du robot mobile.

2.2.1 Procédure

1- Attendre au début du programme qu'un bouton poussoir soit poussé avec la technique d'interrogation (polling) identique au programme précédent.
2- Écrire à l'écran un message d'accuser réception, toujours identique au programme précédent.
3- Insérer un délai d'approximativement de 3-4 secondes, en milliseconde, avant d'exécuter le reste du programme.

   Si durant le délai de 3-4 secondes :
   a. en mode simulation, un utilisateur a appuyé sur une des clefs du clavier ou
   b. en mode réel, un utilisateur a appuyé sur le bouton poussoir PP1 de la carte d'évaluation,

   il faudra afficher sur l'afficheur LCD (simulé ou réel) les valeurs de voltage des trois capteurs IRs après le 4 seconde comme suit :

   XX :YY:ZZ

   où XX est la valeur du voltage hexadécimal du capteur gauche, YY est la valeur hexadécimal du voltage du capteur central, et ZZ la valeur hexadécimale du voltage du capteur droit.

Le bouton poussoir ou la clef du clavier devront activer leur propre routine de service d'interruption, laquelle devra mettre un indicateur d'urgence à 0 (une mémoire RAM quelconque).

Si à la sortie du délai de 3-4 secondes, l'indicateur d'urgence est à 0, on procédera à l'affichage des voltages des capteurs sur le LCD pour ensuite terminer le programme, sinon on n'affichera pas les voltages des capteurs sur le LCD et on terminera le programme.

2.2.2 Mode simulation

1- Pour permettre au récepteur de caractère de l'interface de communications sérielles (SCI Serial Communications Interface) de générer une interruption à la réception d'un caractère en provenance du clavier, il suffit de mettre à « 1 » le bit RIE (Receive Interrupt Enable) du registre SCICR2 du SCI.

Dès que le récepteur de caractères sériels recevra un caractère, le SCI rédigera le programme à l'adresse contenue dans son vecteur associé. Ce vecteur se situe à l'adresse **$FFD6**. L'adresse contenue dans le vecteur d'interruption du SCI sera donc l'adresse de service de l'interruption du SCI.

2- La routine de service d'interruption est à toute fin pratique une routine conventionnelle, sauf pour sa fin qui se termine par un RTI (Return from Interrupt) plutôt qu'un RTS (Return from Subroutine).

Cette routine devra :

a. Lire le registre SCISR1 et SCIDRL pour acquitter l'interruption.
b. Activer une variable d'urgence. À titre d'exemple, mettre le chiffre « 0 » dans la variable en mémoire RAM **Urgence**.
c. Désactiver l'interruption du récepteur en remettant le bit RIE du SCICR2 à « 0 ».

3- Pour brancher l'afficheur LCD au simulateur, ajoutez les deux dernières lignes de commandes suivantes au fichier de commande **Simulateur_SetCPU.cmd** ;

```
// At startup the commands written below will be executed
OPEN Terminal 40 40 40 40
OPEN Push_Buttons
PBPORT $0258
OPEN LCD
LCDPORT $0 $1
```

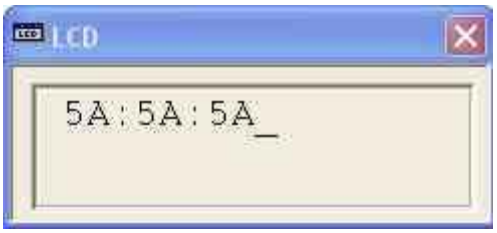


Figure 37. Afficheur LCD en mode simulation

Le PORTB, de 8 bits, à l'adresse $0001 enverra l'octet de contrôle au LCD alors que le PORTA, de 8 bits, à l'adresse $0000 enverra les foncions à exécuter ou les caractères ASCII à afficher au LCD.

Les ports d'entrées/sorties PORTA et PORTB devront être initialisés en mode sortie, c'est à dire, mettre les registres DDRA et DDRB à $FF (Data Direction Register A or B).

4- Mode d'utilisation du LCD en mode simulation ou réel

Le LCD en mode simulation (ou réel) peut afficher une ou deux lignes de 16 caractères avec ou sans curseur à partir de ses deux registres de contrôle branchés sur un port de contrôle et un port de data comme le montre la figure 42.

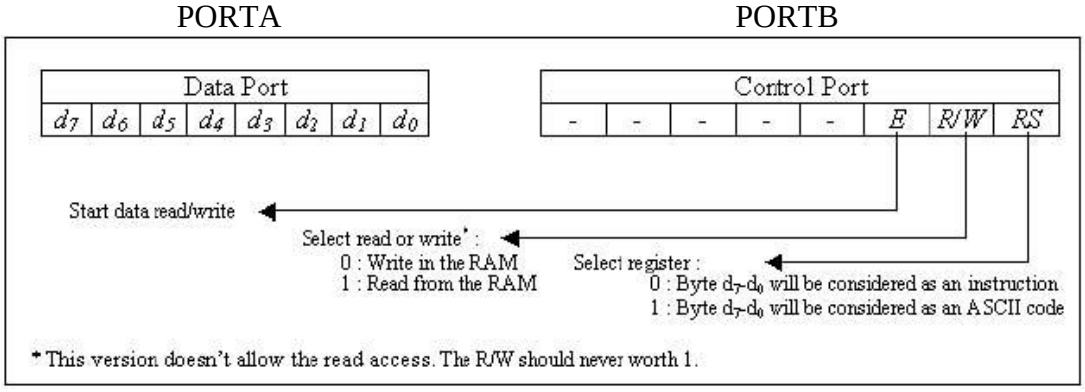

PORTA
PORTB
Data Port
d7 d6 d5 d4 d3 d2 d1 d0
Control Port
- - - - - E R/W RS
Start data read/write
Select read or write* :
0 : Write in the RAM
1 : Read from the RAM
Select register :
0 : Byte d7-d0 will be considered as an instruction
1 : Byte d7-d0 will be considered as an ASCII code
* This version doesn't allow the read access. The R/W should never worth 1.

Figure 38. Registres de contrôle du LCD

Le port de data peut servir à la fois à transmettre une commande ou un caractère ASCII au LCD dépendant de l'état du bit RS du port de contrôle.

Pour envoyer un caractère au LCD, il faut (1) mettre les bits E et RS à « 1 » et R/W à « 0 » dans le port de contrôle, (2) mettre le caractère ASCII dans le port de data, (3) mettre le E bit à 0, (4) attendre 2 ms, et finalement, (5) remettre le E bit du port de contrôle à « 1 ».

Pour envoyer une instruction au LCD, il faut (1) mettre les bits E à « 1 » et les bits RS et R/W à « 0 » dans le port de contrôle, (2) mettre la commande dans le port de data, (3) mettre le E bit à « 0 », (4) attendre 2 à 15ms selon la commande, et finalement, (5) remettre le E bit du port de contrôle à « 1 ».

Les instructions possibles à envoyer au LCD sont les suivantes :

Instruction	Code								Description
	d_7	d_6	D_5	d_4	d_3	d_2	d_1	d_0	
Clear Display	0	0	0	0	0	0	0	1	Erases the display and put AC at 0.
Return Home	0	0	0	0	0	0	1	-	Puts the address 00h into AC and re-init the display.
Entry Mode Set	0	0	0	0	0	1	I/D	-	Fixes the moving direction of the cursor
Display On/Off Control	0	0	0	0	1	D	C	-	Lights on or off the display and shows or not the cursor.
Cursor or Display Shift	0	0	0	1	S/C	R/L	-	-	Moves the cursor and shifts the display.
Set DDRAM Address	1	a_6	A_5	a_4	a_3	a_2	a_1	a_0	Fixes the AC value.
Function Set	0	0	1	DL*	N	-	-	-	Fixes the data exchange width and the line number to display.

Figure 39. Instructions possibles au LCD en mode simulation.

N.B. Pour plus d'information sur les instructions, voir le document "*Manual_Engine_HC12.pdf*" pour le mode simulation ou le fichier "*LCDmanual.pdf*" pour le mode réel.

L'initialisation du LCD demande 8 étapes, voir la figure 40.

L'instruction '*Function Set Command'* doit être envoyer 3 fois de suite pour activer le nombre de bits en transmission et une quatrième fois pour fixer le nombre de lignes à une ou deux.

La figure 39 montre comment initialiser le LCD pour communiquer en 8 bits, à utiliser les deux lignes de l'afficheur, à afficher le curseur et à pousser le AC *(Address Curser)* vers la droite sur réception d'un caractère.

Vous devez respecter les temps d'attentes entre les lignes de codes de l'initialisation et des accès au LCD. Pour ce faire, vous devez concevoir une routine (DELAI) de 1 ms de délai. Cette routine devra accepter le nombre (n * 1 ms) à attendre dans le registre Y. Cette routine pourra donc générer des délais de 1 ms à 65536 ms.

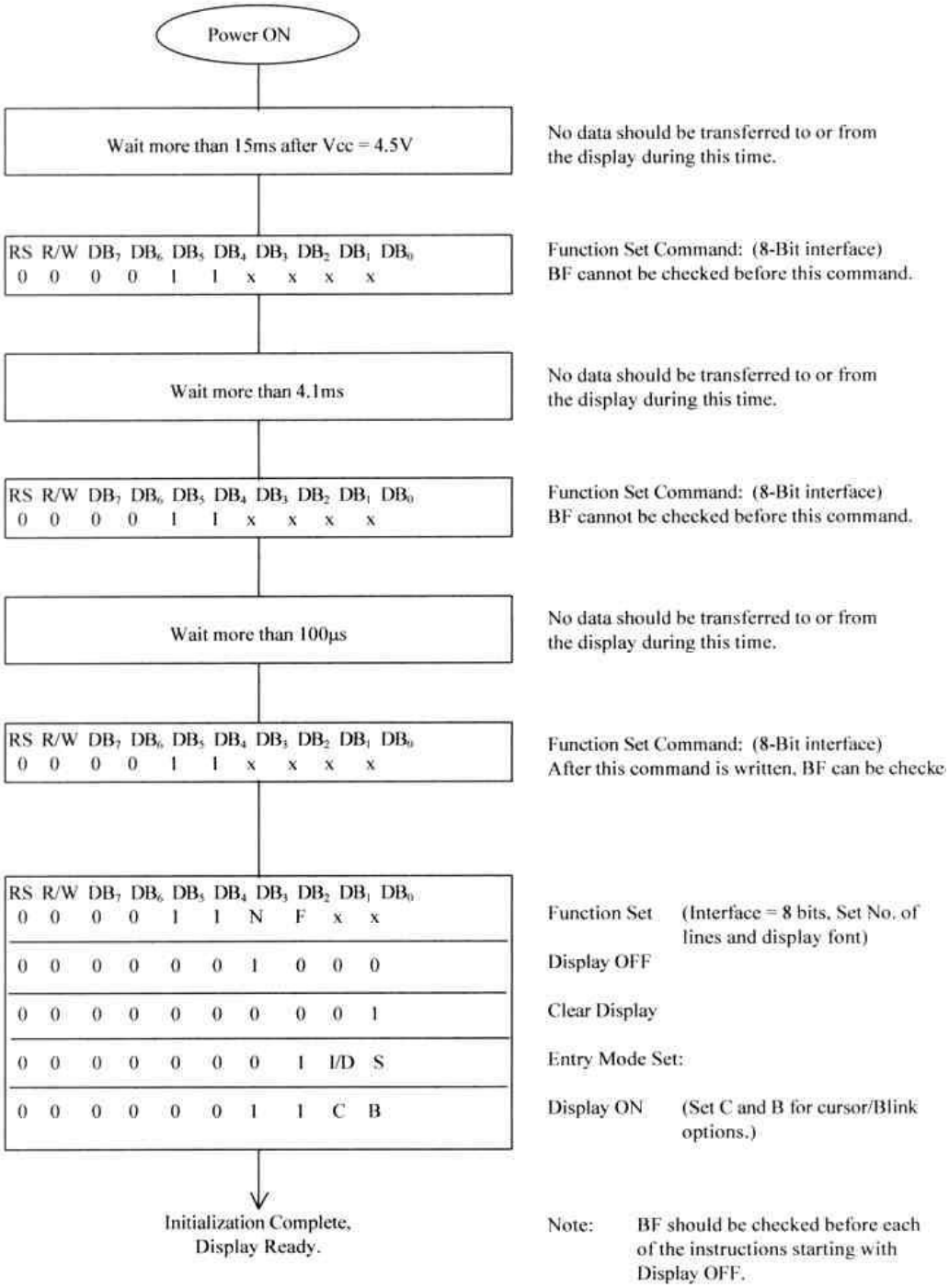

Power ON
Wait more than 15ms after Vcc = 4.5V
No data should be transferred to or from the display during this time.
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 x x x x
Function Set Command: (8-Bit interface)
BF cannot be checked before this command.
Wait more than 4.1ms
No data should be transferred to or from the display during this time.
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 x x x x
Function Set Command: (8-Bit interface)
BF cannot be checked before this command.
Wait more than 100µs
No data should be transferred to or from the display during this time.
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 x x x x
Function Set Command: (8-Bit interface)
After this command is written, BF can be checke
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 N F x x
Function Set (Interface = 8 bits, Set No. of lines and display font)
0 0 0 0 0 0 1 0 0 0
Display OFF
0 0 0 0 0 0 0 0 0 1
Clear Display
0 0 0 0 0 0 0 1 I/D S
Entry Mode Set:
0 0 0 0 0 0 1 1 C B
Display ON (Set C and B for cursor/Blink options.)
Initialization Complete,
Display Ready.
Note: BF should be checked before each of the instructions starting with Display OFF.

Figure 40. Étapes d'initialisations du LCD

2.2.3 Mode réel

1- Pour pouvoir lire l'état du bouton PP0 et permettre de générer une interruption au bouton PP1, (boutons physiquement branchés au port P (PTP) de la carte d'évaluation du HCS12), il suffit de programmer les bits « 0 » et « 1 » du port P comme suit :

   a. en entrées
   b. en mode *Pull-Up*
   c. activés sur un front descendant

   Registres DDRP (*Data Direction Register*) = $00
   PERP (*Pull Device Enable Register*) = $03
   PPSP (*Port P Select Register*) = #$00

2- Pour permettre au port P de générer une interruption lorsque PP1 est appuyé, il suffit de mettre le bit « 1 » du PIEP (*Port P Interrupt Enable Register*) à « 1 » et de mettre le drapeau de PTP1 à « 0 » dans le registre PIFP en y inscrivant un « 1 ».

   a. activer l'interruption
   b. mettre le drapeau d'activation à « 0 »

   Registre PIFP (*Port P Interrupt Flag Register*) = #$02
   PIEP (*Port P Interrupt Enable Register*) = #$02

   Dès que le bouton PP1 sera appuyé, l'interruption rédigera le programme à l'adresse contenue dans son vecteur associé. Ce vecteur se situe à l'adresse **$FF8E**. L'adresse contenue dans le vecteur d'interruption sera donc l'adresse de service de la routine d'interruption du bouton PP1.

   La routine de service d'interruption du PP1 devra :

   a. Activé une variable d'urgence. À titre d'exemple, mettre le chiffre « 0 » dans la variable **Urgence**.
   b. Acquitter l'interruption du port P en remettant le bit « 1 » du PIFP (*Port P Interrupt Flag Register*) à « 0 ». Pour ce faire, il faut écrire un « 1 » à ce bit.

   Ex. :
```
movb    #$02,PIFP       ; Accusé réception
```

   N.B Le bouton PP1 sera utilisé plus tard sur le robot mobile comme bouton d'arrêt d'urgence.

3- Pour afficher le message sur l'afficheur LCD **DMC 16117A** de la compagnie *OPTREX CORPORATION* monté sur le robot mobile, on suit la même procédure qu'en mode

simulation. L’afficheur LCD en mode simulation a été configuré pour répondre comme un LCD **DMC 16117A**.

Figure 41. Afficheur LCD DMC 16117A

2.3 LABORATOIRE #2c Modulation de la durée d'impulsions

Ce laboratoire vous permettra de moduler la durée d'impulsions pour le contrôle de la vitesse des moteurs gauche et droit du robot mobile.

Pour ce faire vous devrez :

a) Moduler la durée d'une impulsion à partir des mécanismes de sorties de signaux sur une comparaison d'horloge (output compare, OCn) où n représente un numéro de 0 à 7 correspondant aux broches de sorties du port T de PT0 à PT7.

b) Compter le nombre d'impulsions qui se rendent aux moteurs avec les captures des temps d'entrées (input capture, ICn) du port T.

c) Contrôler les déplacements du robot en accélération, vitesse constante et décélération en modifiant tout simplement la valeur des compteurs temporels (TC2 et TC3) des Ocn tout en suivant un profil de vitesse préétabli.

d) Configurer les broches du port T.

e) Afficher les secondes sur un terminal.

2.3.1 Introduction

Un des potentiels importants du microcontrôleur 68HCS12 est son module de gestion du temps (TIM). Ce module est utilisé pour la génération, la mesure et la synchronisation des signaux de/du monde externe à travers le port T.

Nous utiliserons les broches PT0 et PT1 du port T en captures d'entrées (IC0 et IC1) et les broches PT3 et PT2 pour générer des signaux en sorties (OC3 et OC2).

2.3.2 Modulation de la durée d'une impulsion

Les contrôleurs attachés aux moteurs du robot mobile, figure 47, acceptent un signal périodique de 16,386ms avec une durée variable pour en contrôler la vitesse. Comme le montre la figure 46, la durée de l'impulsion peut varier de 1,5ms à 2ms pour contrôler une vitesse positive, ou de 1,5ms à 1ms pour contrôler une vitesse négative. Les moteurs seront inertes pour une durée constante d'impulsion de 1,5 ms.

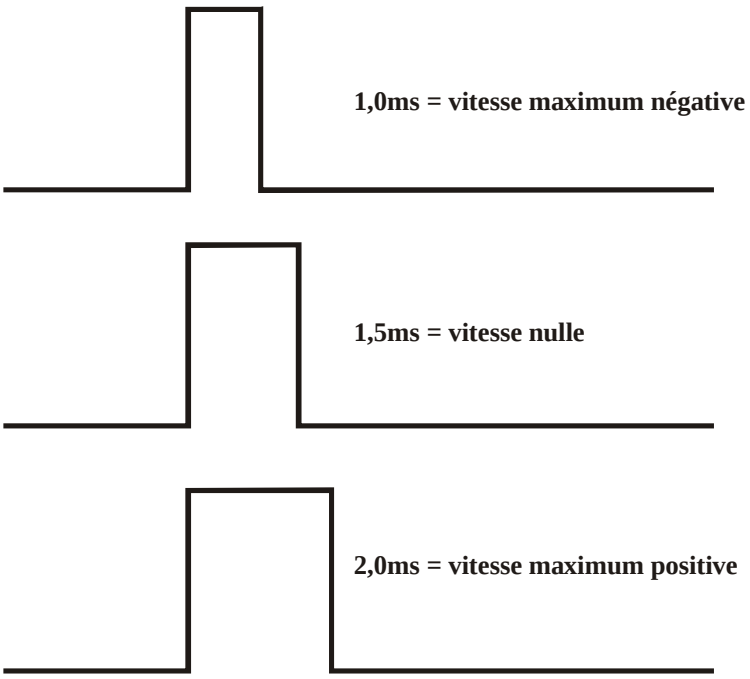


Figure 42. Impulsions envoyées aux contrôleurs des moteurs

Figure 43. Contrôleur et moteur CC du robot mobile

L’ajustement de la vitesse nulle des moteurs se fera en envoyant un signal périodique de 16,384ms avec une durée constante de 1,5ms. Si le moteur tourne dans ces conditions, il suffira d’ajuster le potentiomètre de la figure 48, associé au moteur pour annuler cette vitesse.

Figure 44. Potentiometre pour l'étalonnage nul à 1,5ms

2.3.2.1 Module de l’horloge et des minuteurs de temps (TIM)

Le module TIM fonctionne à partir d’un compteur sans fin de 16 bits, le TCNT branché à ses 8 OCn/ICn. Ce compteur peut compter à des vitesses différentes selon la vitesse de l’horloge qui le contrôle.

Sa vitesse découle du BUSCLK qui est présentement fixer à 8MHz. Comme on désire une période de 16,384ms pour contrôler la vitesse des moteurs, on devra programmer la vitesse de l’horloge du module TIM à 4MHz, ce qui donnera un débordement d’horloge à toutes les 16,384ms, pour un compte de 65536 coups d’horloge.

Pour ce faire, il faut programmer le registre TSCR2 (TMSK2 du livre de cours) du module TIM pour avoir le 8Mhz/2 .

L’activation du module TIM se fait en activant le bit TEN du registre TSCR1 (TSCR du livre de cours).

2.3.2.2 Sorties de signaux sur une comparaison d’horloge (OCn)

Les OCn activent ou désactivent les broches en sorties PT0 à PT7 du 68HCS12 qui ont été programmées en OCn dans le registre TIOS. Chaque activation ou désactivation des broches de sorties se fera quand le compteur programmable de 16 bits (TC2 et TC3) du registre associé aux OCn égalera le compte du compteur TCNT.

L'activation ou la désactivation des broches sur une égalité de leur compteur (TC2 et TC3) au compteur TCNT se fait grâce au registre TCTL1 et TCTL2. Ces deux registres permettent de stipuler un mode de sortie OMn (output mode) et un niveau de sortie OLn (output level) pour chaque broche préalablement programmée en OCn comme le montre le tableau 3.

OMn OLn	Configuration de la sortie des broches
00	Horloge débranché
01	Inversion de la sortie existante
10	Désactiver la sortie
11	Activer la sortie

Tableau 3. Sélection des signaux de sorties des OCn

2.3.2.3 Captures des temps d'entrées (ICn).

Contrairement aux OCn, les ICn s'activent à partir des broches en entrées PT0 à PT7 du 68HCS12 qui ont été programmées en ICn dans le registre TIOS. Chaque changement d'état d'une broche d'entrée enregistrera le contenu du TCNT dans le registre de 16 bits (TC2 et TC3) associés pour calcul de temps ultérieur.

Le changement d'état des broches sur un front montant, sur un front descendant ou sur les deux, se programme avec le registre TCTL3 et TCTL4. Ces deux registres permettent de stipuler un mode d'entrée EDGnB (edgeB) et EDGnA (edgeA) pour chaque broche préalablement programmée en ICn comme le montre le tabeau 4.

EDGnB EDGnA	Configuration de détection sur les broches d'entrées
00	Capture débranchée
01	Front montant
10	Front descendant
11	Les deux fronts

Tableau 4. Sélection des signaux d'entrées des ICn

2.3.2.4 Cas particulier avec le OC7

Il est possible de forcer l'activation ou la désactivation des broches de sorties sur le port T (PT0 à PT6 inclusivement) sur l'égalité du compteur du OC7 (TC7). On donne les broches que l'on veut forcer ainsi dans le registre OC7M (Output Compare 7 Mask) et on donne le data que l'on veut forcer sur ces broches dans le registre OC7D (Output Compare 7 Data)

Pour générer une période constante, on forcera donc l'activation des broches de sorties qui sont branchées aux moteurs sur chaque égalité du compteur du OC7 (TC7) avec le compteur TCNT. Ceci se produira à chaque 65536 comptes. Si on programme le compteur du OC7 (TC7) à #$0000, on forcera les broches de sorties à s'activer à toutes les 16,384ms sur le compte de #$0000, voir la figure suivante.

La désactivation de ces broches se fera par les OC2 et OC3. Ceux-ci désactiveront leur broche quand il y aura égalité de leur compteur (TC2) et (TC3) avec le TCNT.

Une fois les OC2, OC3 et OC7 programmés, nous n'avons plus qu'à modifier les valeurs des compteurs (TC2) et (TC3) pour générer des durées de 1,0ms à 2,0ms, pour contrôler la vitesse des moteurs gauche et droit du robot mobile.

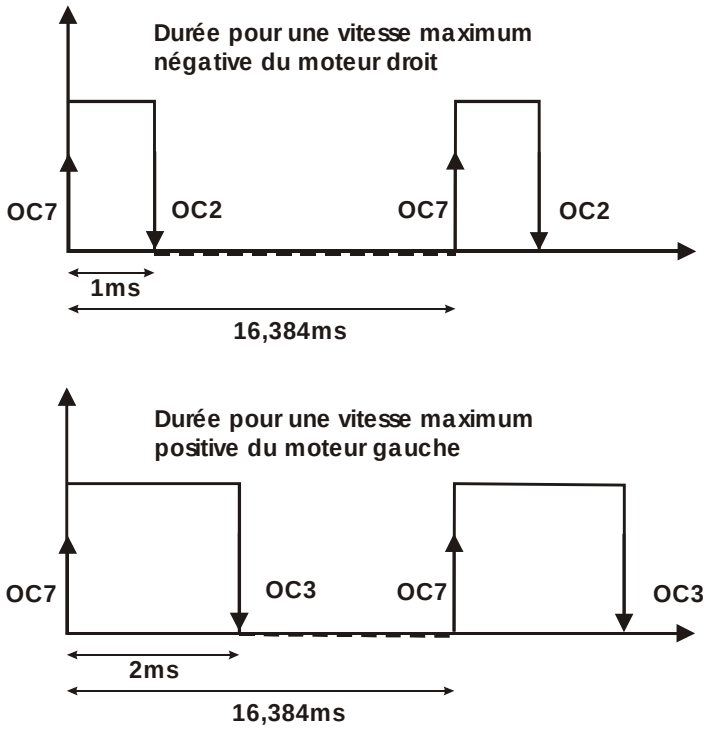


Figure 45. Contrôle de la vitesse des moteurs gauche et droit par la modulation de la durée

2.3.2.5 Profil de vitesse

Dans un premier temps, on vous demande de contrôler la vitesse du robot mobile uniquement en marche avant avec le profile de vitesse suivant.

Comme le montre la figure suivante, vous devrez générer :

1- 10 niveaux d'accélération de 6000 impulsions à 8000 impulsions dans les 5 premières secondes
2- 10 vitesses constantes à 8000 impulsions pour les 5 secondes suivantes
3- 10 niveaux de décélération de 8000 impulsions à 6000 impulsions pour les 5 dernières secondes

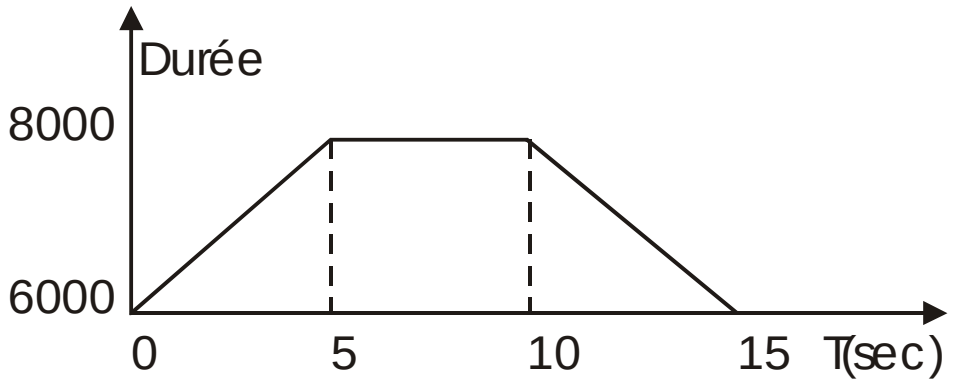


Figure 46. Profil de vitesse pour le contrôle des moteurs en marche avant

Pour ce faire, programmez un tableau de 30 variables de 16 bits représentant les durées associées aux différentes vitesses du moteur droit et 30 autres variables associées aux moteur gauche.

À l'exécution de votre programme en mode réel, mesurez la durée des impulsions ainsi générées sur l'oscilloscope pour vérifier le bon fonctionnement de votre programme.

2.3.2.6 Mesure du nombre d'impulsion arrivant aux moteurs

Les sorties OC3 et OC2 qui contrôlent les moteurs ont été court-circuitées avec les entrées IC0 et IC1 respectivement sur les cartes de développements du laboratoire et des robots mobiles. Il devient donc possible de compter le nombre d'impulsions envoyées aux moteurs par les OCn en les capturant sur les ICn.

Le temps de capture qui sera stocké dans les compteurs (TC0H :TCH0L) et (TC1H :TC1L) associé aux ICn ne nous intéresse pas pour le moment. Ce qui nous intéresse c'est uniquement le nombre d'impulsions capturées par les ICn. Les impulsions arriveront à intervalles réguliers de 16,384ms représentant la période du OC7. On aura donc environ 61 comptes par ICn par seconde.

Le comptage du nombre des impulsions envoyées aux moteurs se fera à l'aide d'interruptions. Les interrupteurs seront les IC0 et IC1 et s'activeront sur le front montant des impulsions.

Pour ce faire, vous devez programmer le registre TCTL4 pour la reconnaissance d'un front montant sur les entrées IC0 et IC1 et permettre les interruptions des entrées IC0 et IC1 dans le registre TIE (Timer Interrupt Enable).

Les vecteurs associés aux interrupteurs IC0 et IC1 sont situés aux adresses **$FFEE** et **$FFEC** dans lesquelles vous devrez mettre les adresses de services des interruptions.

2.3.2.7 Routines de service et programme principale.

Une des routines de service d'interruption devra :

1- compter le nombre d'impulsions qui sont envoyées aux moteurs
2- gérer les secondes à partir de ce compte
3- acquitter la requête d'interruption en déposant un 1 logique sur le drapeau correspondant à l'aide de l'instruction MOVB dans le registre de drapeaux TFLG1.

L'instruction MOVB est utilisée ici pour ne pas affecter les autres drapeaux du registre TFLG1 qui n'auraient pas encore été acquittés.

L'autre routine de service d'interruption n'acquittera, pour l'instant, que sa requête d'interruption en déposant, elle aussi, un 1 logique sur le drapeau correspondant à sa requête.

Finalement, le programme principal affichera à l'écran avec les fonctions DBUG12 les secondes depuis le début du contrôle des moteurs comme suit :

Temps : 01 secondes
Temps : 02 secondes
.
.
.
Temps : 0F secondes

2.4 LABORATOIRE #2d Détection du voltage des capteurs infrarouges

Ce laboratoire vous permettra de mettre en pratique les connaissances acquises dans votre cours sur le convertisseur analogique à numériques (A/N) par approximations successives, intégré au microcontrôleur 68HCS12.

Trois détecteurs d'obstacles par infrarouge (IR), GP2Y0A21YK de la compagnie SHARP, ont été montés sur le robot mobile pour la détection d'un obstacle à l'avant, à droite, ou à gauche.

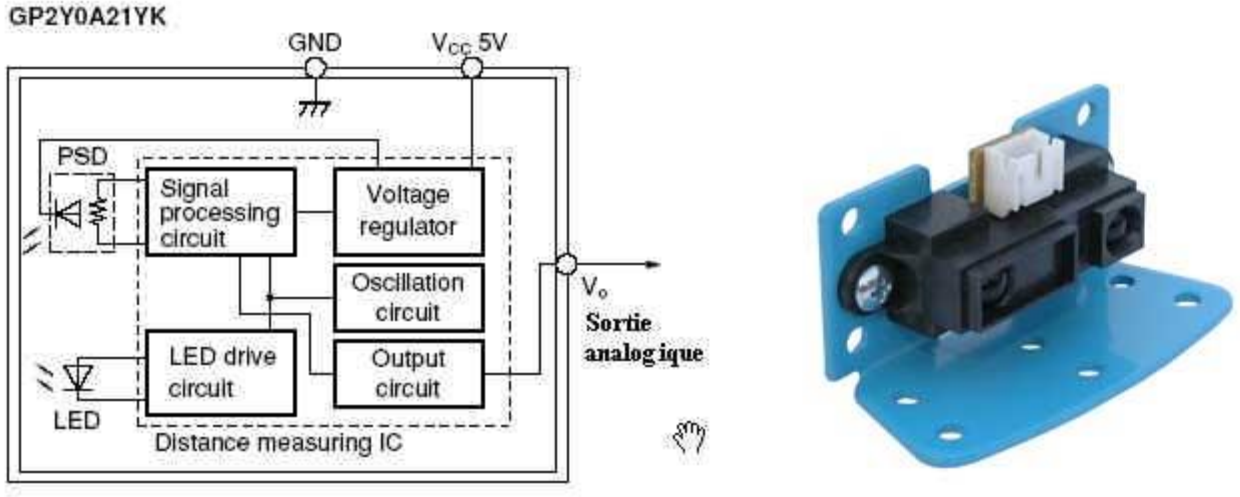


Figure 47. Capteur infrarouge GP2Y0A21YK

Chaque détecteur diffuse une lumière IR à partir d'une diode émettrice de lumière (LED) et capte la quantité d'énergie de lumière réfléchie sur un objet par celle-ci. Cette quantité d'énergie est ensuite traduite par le capteur en voltage à sa sortie analogique. Cette quantité d'énergie est proportionnelle à la distance entre un obstacle et le capteur comme le montre la figure suivante.

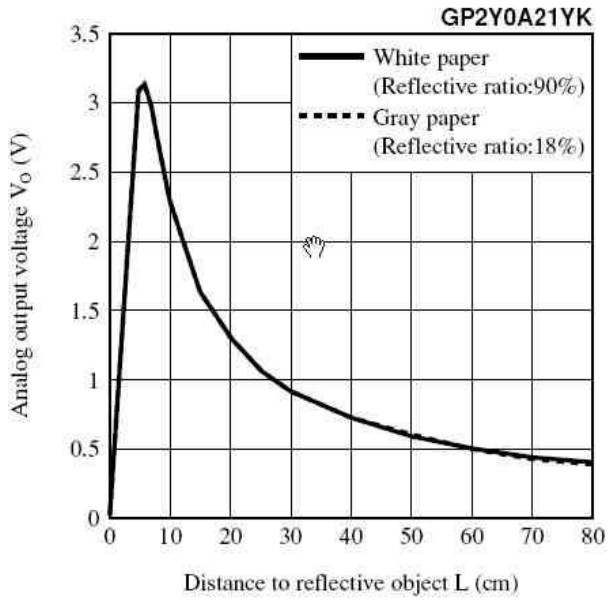


Figure 48. Voltage analogique en fonction de la distance de réflexion d'un objet

Les trois sorties des détecteurs ont été branchées sur les entrées PAD1, PAD2 et PAD3 du convertisseur A/N.

2.4.1 Le convertisseur A/N du 68HCS12

Vous devez donc programmer le convertisseur A/N pour pouvoir convertir trois signaux analogiques en provenance des capteurs. Chaque signal sera traduit sur 8 bits non signé et justifié à droite et sera converti à une vitesse de 2MHz.

Le chiffre obtenu du convertisseur A/N de 8 bits (0 à 256d ou $00 à $FF) correspondra à une valeur comprise entre 0 et 5 volts, basée sur les voltages de référence bas Vrl (*Low Reference Voltage*) et de référence haute Vrh (*High Reference Voltage*). Ces voltages sont déjà branchés sur la carte de développement à 0 volt et 5 volts respectivement.

Pour ce faire vous devez :

1- Activer le convertisseur A/N et stipuler une remise à zéro des drapeaux des canaux convertis, dès la lecture des registres de résultats de ces mêmes canaux.

   Les bits ADPU (*ATD Power Up*) et AFFC (*ATD Fast Flag Clear All*) du registre de contrôle no 2 du convertisseur A/N ATDCTL2 (*ATD Control Register 2*) devront être programmés à 1.

2- Stipuler qu'il y a 3 conversions à faire en programmant les bits S8C, S4C, S2C et S1C (*Conversion Sequence Length*) du registre de contrôle no. 3 du convertisseur A/N ATDCTL3 (*ATD Control Register 3*).

Les bits S1C et S2C du registre ATDCTL3 devront donc être mis à 1.

Notez que la longueur d'une séquence de conversion ne stipule pas dans quels registres de résultats seront déposés les résultats des conversions. Vous verrez un peu plus loin où seront stockés les résultats des conversions.

3- Stipuler une résolution de 8 bits avec une vitesse de conversion de 2MHz. avec 2 coups d'horloge ATDclock pour le '*Sample and Hold*'

Les bits SRES8 (*A/D Resolution Select*) et PRS0 (*ATD Clock Prescaler*) du registre de contrôle no. 4 du convertisseur ATDCTL4 (*ATD Control Register 4*) devront être à 1.

Notez que le BusClock est à 8MHz au départ et que le PRS (*Prescale Values*) est égale à 1 avec les bits PRS4=0, PRS3=0, PRS2=0, PRS1=0 et PRS0=1

ATDclock = ([BusClock] / [PRS + 1]) * 0,5

2MHz = (8MHz / 2) * 0,5

Une conversion prendra 10 coups d'horloge ATDclock donc 1/200KHz = 5 μs. Comme vous avez trois canaux, le temps de conversion total prendra 15 μs.

4- Stipuler des conversions multiples, le numéro du canal de départ et la justification à droite dans les registres de résultats.

Les bits CA (*Analogue Input Channel Select Code*), DJM (*Result Register Data Justification*) et MULT (*Multi-Channel Sample Mode*) du registre de contrôle no. 5 du convertisseur A/N ATDCTL5 (*ATD Control Register 5*) devront être à 1.

Les bits CA, CB et CC déterminent le canal de départ d'une séquence de conversion. Comme vous devez convertir les canaux 1, 2 et 3, il faut débuter la séquence sur le canal no 1.

Attention, les résultats des conversions des canaux 1, 2 et 3 seront stockés dans les registres de résultats ATDDR0L, ATDDR1L et ATDDR2L (*Right Justified, ATD Conversion Result Register, Low Byte*). Les résultats des séquences de conversion commencent toujours à ATDDR0 et non au numéro de canal du début de la séquence.

N.B. **Le fait d'écrire dans le registre ATDCTL5 commande une conversion ou une séquence de conversion**. Ici, vous commandez une conversion A/N sur trois canaux à partir du canal 1. Cette conversion se fera à 2MHz avec un temps de chargement des '*Sample and Hold*' de 2 coups d'horloge. Une fois accomplie (après 15 μs), les réponses seront disponibles en 8 bits dans les registres de résultats ATDDR0L, ATDDR1L et ATDDR2L.

5- Interroger le bit SCF (*Sequence Complete Flag*) du registre de statut 0 du convertisseur A/N ATDSTAT0 (*ATD Status Register 0*). Pour savoir si la conversion des trois canaux est terminée.

Le bit SCF indiquera 1 quand la conversion sera terminée.

Vous pouvez utiliser le code suivant pour tester ce bit.

```
CONVERSION:     BRCLR       ATDSTAT0,#$80,CONVERSION
```

Cette ligne de code va brancher sur elle-même tant et aussi longtemps que les bits qui sont à 1 dans le masque (ici $80 pour le bit SCF), sont à 0 dans l'adresse stipulée (ici ATDSTAT0).

2.4.2 Numérisation des IRs et affichage

À partir du laboratoire 2b, vous devez créer une boucle sans fin de numérisation des trois valeurs de voltages des capteurs.

Cette boucle devra :

1- Commander une numérisation.
2- Tester le bit SCF
3- Sauvegarder les résultats dans trois octets de mémoires (valeur gauche, valeur centre et valeur droite) en ram.
4- Attendre 0,5 seconde
5- Afficher sur l'afficheur LCD en mode réel ces trois valeurs. S'il n'y a pas de LCD disponible pour faire vos tests, afficher vos résultats à l'écran avec les fonctions DBUG12.

XX :YY :ZZ

Où XX est la valeur hexadécimale du voltage du capteur gauche, YY est la valeur hexadécimale du voltage du capteur central, et ZZ la valeur hexadécimale du voltage du capteur droit.

N.B. Pour remettre le pointeur de caractère de l'afficheur LCD au début de la ligne, vous devez envoyer la fonction #$02 (*Return Home*) à l'afficheur et attendre un délai de 4 ms avant d'y afficher de nouveaux caractères.

2.5 Rapport pour le laboratoire #2

2.5.1 Travail à remettre

Pour ce deuxième laboratoire, vous devez remettre :

1- Un imprimé fonctionnel et très bien commenté de vos programmes LAB2b.ASM, LAB2c.ASM et LAB1d.ASM ainsi qu'un imprimé de votre rapport.

N.B. Pour l'imprimé, prenez soin d'utiliser une police à pas fixe (COURRIER) capable de reproduire la plus grande ligne de texte de votre programme sur une seule ligne de l'imprimé. L'impression en mode paysage est acceptée.

2- Une **copie sur MOODLE** de vos programmes LAB2b.ASM LAB2c.ASM et LABd.ASM. ainsi que de votre rapport *.doc.

2.5.2 Barème de correction

Programme (a/b)

Fonctionnalité, incluant une présentation orale au chargé de TPs……..15%
Code structuré et commentaires……..5%
Rapport…….8%
Introduction……...1%
Objectifs……1%
Méthodes……...3%
Discussion…….2%
Conclusion……1%

Programme (c)

Fonctionnalité, incluant une présentation orale au chargé de TPs……..15%
Code structuré et commentaires……..5%
Rapport…….8%
Introduction……...1%
Objectifs……1%
Méthodes……...3%
Discussion…….2%
Conclusion……1%

Programme (d)

Fonctionnalité, incluant une présentation orale au chargé de TPs……..15%
Code structuré et commentaires……..5%
Rapport…….8%
Introduction……...1%
Objectifs……1%
Méthodes……...3%
Discussion…….2%
Conclusion……1%

Présentation globale du rapport……10%
Présentation globale du code.……6%

Fin du laboratoire #2

---

3 LABORATOIRE #3 Navigation temps réel du robot mobile

Jusqu'à présent, vous avez étudié et mis en place : (1) un contrôleur à logique flou, (2) une détection de bouton poussoir par interrogations ou par interruptions, (3) une communication vers un afficheur LCD, (4) une modulation d'impulsion pour le contrôle de la vitesse des moteurs et (5) un détecteur de proximité pour le robot mobile.

Après la mise au point d'un contrôle de direction, ce laboratoire vous permettra finalement de réaliser le robot mobile final.

3.1 LABORATOIRE #3a Contrôle de la direction du robot mobile

À partir de la modulation de la durée d'une impulsion, ce laboratoire vous permettra de contrôler l'orientation du robot mobile.

Pour ce faire, vous devrez :

a) Utiliser les mécanismes des signaux de sortie sur une comparaison d'horloge (output compare, OCn) pour contrôler la vitesse des moteurs droit et gauche.

b) Utiliser les mécanismes de la capture des temps d'entrées (input capture, ICn) pour compter le nombre d'impulsions envoyées à un moteur.

c) Changer la vitesse des moteurs en ajustant la durée des impulsions qui entrent dans le servomoteur de chacun des moteurs.

d) Configurer le port T comme dans le labo2b.

3.1.1 Introduction

Vous devez concevoir une routine de direction pour le robot mobile. Cette routine utilisera les services que vous offre le module de la gestion du temps TIM préalablement expérimenté dans le laboratoire précédent.

Cette routine devra être en mesure de déplacer le robot en ligne droite, ainsi que de le faire tourner avec un angle donné.

Vous devez choisir **un ou plusieurs** types de braquage pour contrôler la direction du robot.

1- Braquage court en inversant le sens de rotation des moteurs.

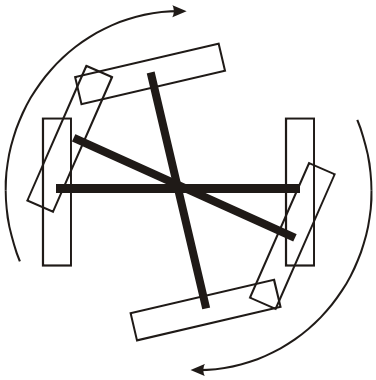


Figure 49. braquage court

2- Braquage moyen en stoppant une roue et en actionnant l'autre.

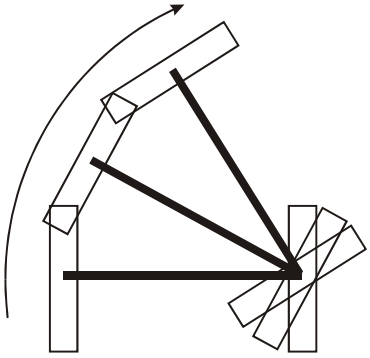


Figure 50. Braquage moyen

3- Braquage long en calculant un rayon de braquage en donnant des vitesses différentes aux deux moteurs

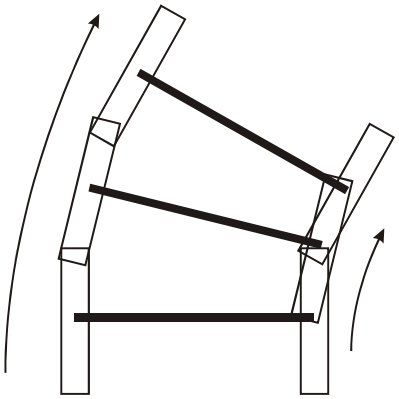


Figure 51. Braquage long

3.1.2 Calcul des braquages

La routine devra être en mesure d'accepter une vitesse pour les moteurs de droite et de gauche ainsi que le temps requis pour le braquage (angle de braquage).

Avant l'appel de la routine, vous devrez :

1- déterminer la durée de l'impulsion à envoyer aux servomoteurs pour en contrôler la vitesse,
2- déterminer le nombre d'impulsions à envoyer aux servomoteurs pour contrôler le temps de braquage.

La vitesse des moteurs sera toujours constante dans un braquage. Vous pouvez choisir dans le tableau d'exemples suivant la vitesse et le temps de braquage que vous désirez. Bien que les valeurs du tableau soient de 6000 à 8000 pour des durées de 1,5 à 2,0ms, vous pouvez l'interpréter de 6000 à 4000 pour des durées de 1,5 à 1,0ms.

Tableau 5. Exemples de vitesses et de braquages moyens pour une rotation de 90 degrés

Valeur TCn	RPM*2	Durée de l'impulsion (ms)*2	Vitesse (mm/S)	Temps pour tourner de 90º (mS)	Nombre d'impulsions requises
6030	0,00	1,508	0,00		
6040	3,75	1,510	14,58	8000,00	489,6
6050	5,00	1,520	19,44	6000,00	367,2
6060	6,50	1,515	25,27	4615,38	282,5
6070	8,00	1,518	31,10	3750,00	229,5
6080	9,25	1,520	35,96	3243,24	198,5
6090	10,50	1,523	40,82	2857,14	174,9
6100	11,80	1,533	45,87	2542,37	155,6
6110	13,00	1,528	50,53	2307,69	141,2
6120	14,50	1,530	56,36	2068,97	126,6
6130	15,75	1,533	61,22	1904,76	116,6
6140	17,00	1,535	66,08	1764,71	108,0
6180	18,75	1,545	72,88	1600,00	97,9
6230	21,50	1,558	83,57	1395,35	85,4
6280	29,00	1,570	112,73	1034,48	63,3
6330	35,70	1,583	138,77	840,34	51,4
6380	41,00	1,595	159,37	731,71	44,8
6430	46,10	1,608	179,20	650,76	39,8
6480	50,50	1,620	196,30	594,06	36,4
6530	59,30	1,633	230,51	505,90	31,0
6580	63,50	1,645	246,84	472,44	28,9

6630	66,50	1,658	258,50	451,13	27,6
6680	69,50	1,670	270,16	431,65	26,4
6730	75,40	1,683	293,09	397,88	24,3
6780	77,30	1,695	300,48	388,10	23,8
6830	80,30	1,708	312,14	373,60	22,9
6880	81,30	1,720	316,03	369,00	22,6
6930	83,30	1,733	323,80	360,14	22,0
6980	84,50	1,745	328,47	355,03	21,7
7030	84,70	1,758	329,24	354,19	21,7
7080	85,00	1,770	330,41	352,94	21,6
7130	86,00	1,783	334,30	348,84	21,3
7180	86,70	1,795	337,02	346,02	21,2
7230	86,40	1,808	335,85	347,22	21,2
7280	87,70	1,820	340,90	342,08	20,9
7330	89,00	1,833	345,96	337,08	20,6
7380	89,10	1,845	346,35	336,70	20,6
7430	90,00	1,858	349,85	333,33	20,4
7480	90,00	1,870	349,85	333,33	20,4
7530	90,00	1,883	349,85	333,33	20,4

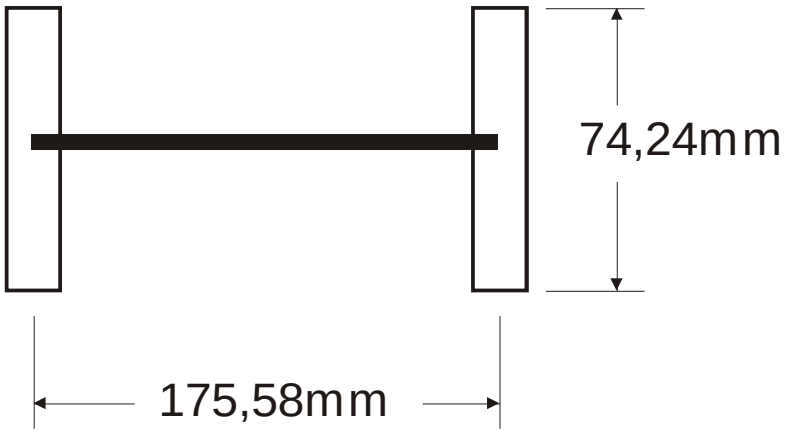


Figure 52. Mesures des roues du robot mobile

3.1.3 Exemple de calcul d'un braquage court

3.1.3.1 Choix de la vitesse de braquage

La force des moteurs est tel qu'elle permet de choisir une des vitesses du tableau précédent sans égard à la vitesse actuelle du moteur. Autrement dit, vous pouvez passer du maximum de la vitesse arrière au maximum de la vitesse avant (**essayez quand même de rester doux**).

Une vitesse de rotation de 90 degrés d'environ 0,6 à 0,7 secondes semble raisonnable. Dans cet exemple nous voulons un braquage court, il s'agit donc de trouver le braquage moyen correspondant dans le tableau 5.

3.1.4 Calcul de l'angle de braquage

Pour l'exemple, nous avons choisi une vitesse de rotation de 90 degrés en 1,39535 seconde en braquage moyen. Ceci donnera en braquage court, un temps de rotation de 90 degrés en 0,698 seconde.

Comme le montre le tableau 5, la durée requise des OC3 et OC2 pour une vitesse de braquage moyen est de 1,558 ms. Pour générer cette durée, nous donnons un compte de 6230d aux compteurs TC2 et TC3 du TIM. Comme les moteurs sont inversés, un par rapport à l'autre, il s'agit d'envoyer la même durée à chaque servomoteur pour pouvoir exécuter un braquage court.

En braquage moyen, il faut générer 85,4 impulsions pour une rotation de 90 degrés, alors en braquage court, nous aurons besoin de deux fois moins soit, 42,7 impulsions.

Avec les erreurs de précision des moteurs et des mesures, nous pouvons nous permettre de faire l'approximation suivante :

2 degrés de braquage par impulsion

Reste à donner à la routine de service d'interruption les trois informations suivantes :

1- Vitesse du moteur droit.
2- Vitesse du moteur gauche.
3- Angle de braquage ou nombre d'impulsions.

Une durée d'impulsion supérieure à 1.5ms aux servomoteurs produira un virage à droite, alors qu'une durée inférieure à 1,5ms produira un virage à gauche.

3.2 LABORATOIRE #3b Robot mobile

Vous avez maintenant tous les éléments pour la mise en marche finale de votre robot mobile. Cette mise en marche se fera en deux temps.

Dans un premier temps, vous étalonnez votre robot pour la position arrêt et vous compensez la vitesse des moteurs pour la marche avant en ligne droite.

Dans un deuxième temps vous ajustez vos règles d'inférence du contrôleur flou. Généralement, l'ajustement de ces règles se fait par un expert, ici l'expert c'est vous.

Vous êtes entièrement libre de réaliser toutes les étapes du robot mobile comme bon il vous semble.

3.2.1.1 Point mort

1- Envoyez un signal d'une durée de 1,5ms aux moteurs droit et gauche.
2- Ajustez les potentiomètres des moteurs jusqu'à l'arrêt complet des roues.

3.2.2 Étalonnage

Les étapes d'étalonnages peuvent se faire dans votre programme.

Une bonne façon de faire, est de programmer une routine de service d'interruption d'étalonnage, amorcée par un des boutons poussoir de la carte d'évaluation.

3.2.2.1 Alignement pour la marche avant

1- À vitesse constante additionnez ou soustrayez une constante à vos valeurs de TC2 et TC3.
2- Ajustez cette constante jusqu'à ce que le robot se déplace en ligne droite

Il serait judicieux de faire cette alignement dynamiquement en appuyant sur un des boutons poussoir afin d'occasionner un incrément ou un décrément dans une des constantes compensatrices pour les TC2 et TC3.

3.2.3 Suggestions et approches.

1- Utilisez un bouton de mise en marche et d'arrêt.
2- À la mise en marche, appliquez un profil d'accélération sans asservissement.

3- Restez à vitesse constante jusqu'à la mise en arrêt.
4- À la mise à l'arrêt, appliquez un profil de décélération sans asservissement et attendre une autre commande.

3.2.3.1 Routine de service d'interruption des moteurs

La routine de service d'interruption des moteurs pourrait, à la fois, servir à l'accélération, la décélération, au braquage à vitesse constante et à la marche avant à vitesse constante.

Pour se faire, il suffit d'utiliser une variable d'identification pour l'un des modes d'utilisation. Les modes accélération et décélération seront fixés par le programme principal alors que les modes de vitesse constante et arrêt seront fixés par la routine de service d'interruption elle-même.

3.2.3.2 Routine sans fin du programme principale

La routine sans fin du programme principal devrait être en mesure de :

Si à l'arrêt
1- Attendre le bouton de mise en marche ou d'étalonnage
2- Indiquer le mode actif
3- Retourner au début de la routine sans fin

Si en accélération
1- Attendre la marche avant
2- Retourner au début de la routine sans fin

Si en marche avant
1- Numériser les voltages des trois capteurs de proximité.
2- Sauvegarder ces valeurs dans les trois variables d'entrées du contrôleur flou.
3- Amorcer une autre conversion.
4- Évaluer la valeur de braquage avec le contrôleur flou.
5- Sauvegarder la valeur de braquage pour la routine de service d'interruption
6- Attendre 250ms
7- Afficher les valeurs des capteurs de proximité sur l'afficheur LCD
8- Retourner au début de la routine sans fin

Si en décélération
1- Attendre l'arrêt
2- Retourner au début de la routine sans fin

3.2.3.3 Routine de service d'interruption pour les pare-chocs

Enfin, Comme les capteurs de proximité ne peuvent voir les petits obstacles, vous devrez mettre au point une routine de service d'interruption pour les pare-chocs. Cette routine devrait 1) mettre le robot en marche arrière pour un temps donné et 2) braquer de X degrés avant de retourner en marche avant

N.B. Les pare-chocs sont branchés sur le port P à PP2 et PP3. Il n'existe qu'un vecteur d'interruption pour les huit broches du port P. Votre routine de service pourra déterminer, si elle en a besoin, quelle broche a générée l'interruption en interrogeant les drapeaux du port P.

3.3 Rapport pour le laboratoire #3

3.3.1 Travail à remettre et barème de correction

Pour ce dernier laboratoire, vous devez :

1- Remettre un rapport expliquant chaque étape de votre programme……..…….15%
   Votre utilisation de la logique floue
   Méthode de braquage
   Méthode utilisé si le robot est en buté
   Méthode d'étalonnage
   Particularités et finesses
2- Remettre un manuel d'utilisateur…….10%
3- Remettre un imprimé fonctionnel et très bien commenté
   de votre programme LAB3b.ASM.……..……10%
4- Faire une démonstration fonctionnelle des performances de votre robot mobile
   Fonctionnalité……..……55%
   Démonstration…….10%

N.B. Pour l'imprimé, prenez soin d'utiliser une police à pas fixe (COURRIER) capable de reproduire la plus grande ligne de texte de votre programme sur une seule ligne de l'imprimé. L'impression en mode paysage est acceptée.

5- Remettre une **copie électronique dans MOODLE** de votre programme *.asm, de votre manuel d'utilisateur *.docx et de votre rapport *.docx.

Fin du laboratoire #3

Appendice A

Syntaxe de compilation pour le langage micro assembleur de Motorola

Mode d'adressage		Format du champ de l'opérande
1)	**Inhérent**	Pas d'opérande
2)	**Immédiat**	#opérande (8/16 bits entier)
3)	**Direct**	opérande (8 bits adresse)
4)	**Étendu**	opérande (16 bits adresse)
5)	**Relatif**	étiquette (déplacement relatif 8/16 bits signé)
6)	**Offset** 5 bits et **index** R	opérande5bits,R (R = X, Y, SP ou PC)
7)	**Offset** 3 bits **pré-décrément** et **index** -R	opérande3bits,-R (R = X, Y ou SP)
8)	**Offset** 3 bits **pré-incrément** et **index** +R	opérande3bits,+R (R = X, Y ou SP)
9)	**Offset** 3 bits **post-décrément** et **index** -R	opérande3bits,R- (R = X, Y ou SP)
10)	**Offset** 3 bits **post-incrément** et **index** -R	opérande3bits,R+ (R = X, Y ou SP)
11)	**Offset** A ou B 8 bits ou D 16 bits et **index** -R	opérandeABouD,R (R = X, Y, SP ou PC)
12)	**Offset** 9 bits signés et **index** R	opérande9bits,R (R = X, Y, SP ou PC)
13)	**Offset** 16 bits et **index** R signés ou pas (sans effet dans 64 K octets)	opérande16bits,R (R = X, Y, SP ou PC)
14)	**Offset** 16 bits et **index** R **indirect** pointent l'argument	[opérade16bits,R] (R = X, Y, SP ou PC)
15)	**Offset** D et **index** R **indirect** pointent l'argument	[opéradeD,R] (R = X, Y, SP ou PC)

Où : - A et B sont des accumulateurs de 8 bits.
- D est la concaténation des accumulateurs A:B et représente un accumulateur de 16 bits.

Exemples de mode d'adressage :

Mode d'adressage		
1)	Inhérent	INX Incrémente de un le registre d'indexation X.
2)	Immédiat	LDX #$1234

Le mode immédiat stipule que l'argument est immédiatement après l'instruction. Après l'exécution, X = $1234

3) Direct

```
LDAB   $0F
```

L'opérande de 8 bits pointe l'argument dans la partie basse de la mémoire entre l'adresse $0000 et $00FF. Après l'exécution de cette instruction, B = au contenu de l'adresse $000F

4) Étendu

```
STD    $F03B
```

L'opérande de 16 bits pointe l'argument dans une plage mémoire de 64 K octets. Après l'exécution D = au contenu de l'adresse $F03B et $F03C parce que D est un 16 bits

5) Relatif

```
BEQ    TX_FIN
```

Le compilateur calcule la distance en octets qui sépare le PC de l'étiquette TX_FIN et la place dans l'argument de BEQ. Si le test est positif, on repositionne le PC relativement au test.

6) Offset 5 bits et index R

```
LDAB   0,X
STAB   -8,Y
```

L'opérande de 5 bits est additionné au registre d'indexation pour former l'adresse de l'argument.

7) Offset 3 bits et index -R Pré-décrément

```
STAA   1,-SP
```

L'opérande de 3 bits est pré-soustrait au registre d'indexation SP pour former l'adresse de l'argument.

8) Offset 3 bits et index +R Pré-incrément

```
LDAB   2,+X
```

L'opérande de 3 bits est pré-additionné au registre d'indexation X pour former l'adresse de l'argument.

9) Offset 3 bits et index R- Post-décrément

```
LDAB   1,X-
```

L'opérande de 3 bits est post-soustrait au registre d'indexation X après avoir charger l'argument à l'adresse X dans l'accumulateur A.

10) Offset 5 bits et index R+ LDAB 3,X+

L'opérande de 3 bits est post-additionné au registre d'indexation X après avoir charger l'argument à l'adresse X dans l'accumulateur B.

11) Offset A ou B et index R ou offset D et index R LDAB D,X

L'accumulateur D est additionné à X pour former l'adresse de l'argument.

12) Offset 9 bits et index R LDAB $FF,X

L'opérande de 9 bits est additionné à X pour former l'adresse de l'argument.

13) Offset 16 bits et index R signé ou pas LDAB $1234,X

L'opérande $1234 est additionné à X pour former l'adresse de l'argument.

14) Offset 16 bits et index R indirect pointent l'argument LDAB [10,X]

L'opérande 10 est additionné à X et pointe l'adresse de l'adresse de l'argument qui sera chargé dans l'accumulateur B.

15) Offset D et index R indirect pointent l'argument LDAB [D,X]

L'opérande D est additionné à X et pointe l'adresse de l'adresse de l'argument qui sera chargé dans l'accumulateur B.

Appendice B

Directives de compilation pour le langage assembleur de Motorola

ABSENTRY	Stipule le point d'entrée en adresse absolue pour le dévermineur.
XDEF	Stipule le point d'entrée si utilisé comme référence externe à un autre module.
;	stipule que ce qui suit est en commentaire.
ORG	(ORIGIN) stipule au compilateur de commencer à compiler à l'adresse indiquée. On peut mettre autant de ORG que l'on veut dans un programme.
EVEN	Force la compilation à une adresse paire.
$	stipule au compilateur que le nombre qui suit est en hexadécimal.
%	stipule au compilateur que le nombre qui suit est en binaire.
@	stipule au compilateur que le nombre qui suit est en octal.
DC.W	(DEFINE CONSTANT SIZE = WORD) mettre la constante suivante à l'adresse où le compilateur est rendu dans sa compilation.
FDB	(FORM DOUBLE-BYTE) Synonyme à DC.W .
DC.B	(DEFINE CONSTANT SIZE = BYTE) mettre la constante suivante à l'adresse où le compilateur est rendu dans sa compilation. Synonyme à DC.
FCB	(FORM CONSTANT BYTE) Synonyme à DC.B.
INCLUDE	colle le fichier texte attaché à la directive à tout endroit de votre programme où la directive est invoquée.
LIST	à partir de la directive, mettre le résultat de la compilation avec le texte du programme dans un fichier *.lst .
NOLIST	à partir de cette directive, on désactive la directive LIST. Noter que LIST et NOLIST peuvent être employées aussi souvent que l'on veut.
MACRO	marque le début d'édition d'un macro.
\@	compteur d'appels d'une macro.

ENDM	marque la fin de l'édition d'une macro.
END	Fin du texte source pour le compilateur.
EQU	Assigne une valeur permanente à une étiquette.
SET	Assigne une valeur temporaire à une constante.
DS.B *n*	Réserve *n* octets en mémoire. Synonyme à RMB.
DS.W *n*	Réserve *n* mots en mémoire.
IF <cond.>	Si condition est vrai on compile le code assembleur suivant le IF Relation = "=" | "!=" | " >=" | ">" | "<=" | "<" | "<>"
ELSE	Sinon on copile le code assembleur suivant le ELSE
ENDIF	Fin du IF ELSE

Example:

```
Try: EQU 0

    IF  Try != 0
      LDAA   #103
    ELSE
      LDAA   #0
    ENDIF
```

Appendice C

Standard du format S_RECORD de MOTOROLA

The Motorola S-Record format encodes programs or data files in an ASCII format. S-records are character strings of five fields: record type, record length, memory address, data and checksum. Each byte of binary data is encoded as a two-character hexadecimal number: the first character represents the high-order four bits and the second the Loin-order four bits of the byte.

TYPE LENGTH ADDRESS DATA CHECKSUM
Fields in a Motorola S-Record record

Field	Length (Bytes)	Description
TYPE	2	This field indicates the start of a record and identifies the record type as folLoins: S0 = Header record S1 = Data record for 16-bit addresses S2 = Data record for 24-bit addresses S3 = Data record for 32-bit addresses S5 = Count record S7 = Terminator record for 32-bit addresses S8 = Terminator record for 24-bit addresses S9 = Terminator record for 26-bit addresses
LENGTH	2	This field specifies the record length which includes the address, data and checksum fields. The 8-bit record length value is converted to two ASCII characters. The first character represents the most significant nibble.
ADDRESS	4, 6 or 8	This field contains the load address.
DATA	0 - n	This field contains the data bytes.
CHECKSUM	2	The last two bytes in the data record contain the record checksum. The checksum is the sum of the record length, load address and data fields. This sum is then complemented (1's complement) and converted to two ASCII characters.

Description of the fields of a Motorola S-Record record

Appendice D

Commandes pour le dévermineur

HI-WAVE:

VER	Shows the version of all loaded commands
AUTOSIZE	Selects window sizing mode
OPENIO	Loads an IO simulation component
OPENPROJECT	Opens an existing project
OPEN	Opens a component window
SET	Loads a target component
LOAD	Loads an application (Code & Symbols)
LOADCODE	Loads an application (Code only)
LOADSYMBOLS	Loads an application (Symbols only)
FONT	Changes font in component windows
BCKCOLOR	Changes background color of component windows
SLAY	Saves the layout and options of all components
ACTIVATE	Activates a window component (inout focus)
CLOSE	Closes a component window
SYSTEM	Executes an external application
EXIT	Terminates this application
RESET	Resets the target MCU
HELP	Displays available commands

HI-WAVE Engine:

LF	Opens a log file
NOLF	Closes a log file
CR	Records all commands to a file
NOCR	Stops recording commands to a file
LOG	Specifies items to be logged
BS	Sets breakpoint
SAVEBP	Saves breakpoints into a file
STEPINTO	Step Into
STEPOUT	Step Out
STEPOVER	Step Over
RESTART	Restart execution
DDEPROTOCOL	DDE Protocol options
BC	Clears breakpoint
BD	Displays breakpoint(s)
G	Starts execution (Go)
GO	Starts execution (Go)
S	Stops execution (Halt)
STOP	Stops execution (Halt)
P	Executes an instruction (Flat step)
T	Executes CPU instruction(s)

WB	Writes byte(s) into target memory
WW	Writes word(s) into target memory (2 bytes)
WL	Writes long(s) into target memory (4 bytes)
MS	Writes byte(s) into target memory (same as WB)
RD	Displays registers
RS	Sets registers
MEM	Displays Memory map
DASM	Disassembles target memory
DB	Displays byte(s) from target memory
DW	Displays word(s) from target memory (2 bytes)
DL	Displays long(s) from target memory (4 bytes)
CD	Displays/Changes directory
E	Evaluates an expression and displays its result
A	Evaluates an expression and assigns its result to an existing variable
PRINTF	Display a string on the window using printf like format
FPRINTF	Write a string to a file using fprintf like format
NB	Changes or displays the default number base for the value of expressions
LS	Lists also global variables and procedures of the loaded application
SREC	Loads of Motorola S-records from a specified file
SAVE	Saves a specified block of memory to a specified file in Motorola S-record format
PAUSETEST	Displays a modal message box for testing purpose
TESTBOX	Displays a modal message box with a given string
REGFILE	Loads the I/O register descriptions from a 'register file'
REGBASE	Sets the base address of the on-chip I/O registers
DEFINE	Defines a symbol and associates a value
UNDEF	Removes a symbol definition
RETURN	Terminates the current command processing level
GOTO	Goes to the line folLoining the label
GOTOIF	Goes to the line folLoining the label if condition is TRUE
WHILE	Executes commands as long as the condition is true
FOR	Executes commands up to a predefined number of times
REPEAT	Executes commands until a certain condition is true
IF	Executes different command sections depending on the conditions
FOCUS	Assignes a component as the destination for all subsequent commands
WAIT	Waits by time tenths of a second
AT	Executes the next command at time (in ms)
CF	Executes commands in the specified command file
CALL	Executes commands in the specified command file

Source:

SPC	Highlights the statement corresponding to the code address
SMEM	Highlights the statements corresponding to the code address range
SMOD	Loads the corresponding module's source text
SPROC	Highlights the statement of the procedure that is in the procedure chain
FOLD	Hides source text for clearness at program block level
UNFOLD	Exhibits the contents of folded source text blocks

SLINE	Displays the line
FINDPROC	Find the Procedure
FIND	Searches an arbitrary pattern in the currently loaded source file
ATTRIBUTES	Sets up the display

Assembly:

SPC	Shows the specified address
SMEM	Shows a memory range
ATTRIBUTES	Sets up the display

Procedure:

ATTRIBUTES	Sets up the display

Register:

ATTRIBUTES	Sets up the display

Memory:

SPC	Shows the address given as argument
SMEM	Shows the memory range given as argument
SMOD	Shows the first global variable of the module
FILL	Fills a memory range with the given value
UPDATERATE	Sets the update rate
ATTRIBUTES	Sets up the display
COPYMEM	Copies a memory range to a specified location
SEARCHPATTERN	Search a pattern in memory

Data:

SPROC	Displays local or global variables of the procedure given as parameter
ADDXPR	Adds a new expression in the data component
PTRARRAY	Switches on or off the pointer as array displaying
SMOD	Displays global variables of the module given as parameter
ZOOM	Exhibits the member fields of structures by 'diving' into the structure
UPDATERATE	Sets the update rate of the data component
DUMP	Dumps the content of the data component to the command line
ATTRIBUTES	Sets up the display

Command:

ATTRIBUTES	Sets up the display

Simulator:

TRUETIME	Returns the true time in microseconds
OSCFREQUENCY	Displays/sets the oscillator frequency
SHOWCYCLES	Shows the number of MCU cycles
RESETCYCLES	Sets the internal MCU cycle counter to typed value
RESETSTAT	Resets the statistical data
RESETRAM	Resets the RAM to 'undefined'
RESETMEM	Resets specified memory to 'undefined'

LOADMEM	Load a memory configuration file
SETCPU	Load the CPU awareness
CMDFILE	Specify a command file state and full name
TRIGGER	Define a range to be recorded in the trace

HC12 Simulator:

HCS12_SUPPORT	enables HCS12 specific mode for MOVB MOVW instructions
BGND_CYCLES	alLoins to adopt the simulator's clock cycles for BGND advise
HALT_ON_TRAP	stop on call to exception handler
PSMODE	controls power save modes (STOP or WAIT mode)

FCS:

WARNING_SETUP	Set FCS behavior for warning messages
MESSAGE_HIDE_ID	Hide a message with specific ID
MESSAGE_SHOW_ID	Show a message with specific ID
MESSAGE_HIDE_RESET	All previously hidden messages to be displayed again

FCS:

ATD0_SETPAD	Set PAD with the given voltage in floating point format

N.B. La syntaxe complète de ces commandes se trouve dans le fichier
Manual_Engine_HC12.pdf

Appendice E

Table de caractères ASCII

Utilisez "ALT" sur votre clavier et le code "dec"

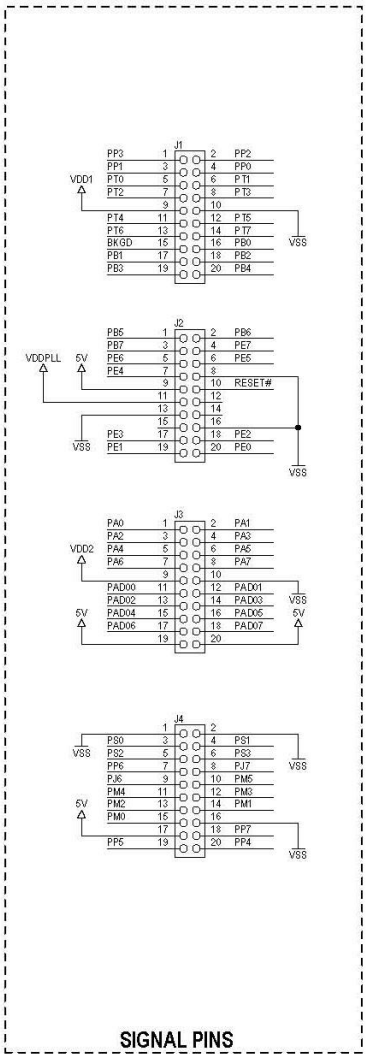
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	|	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

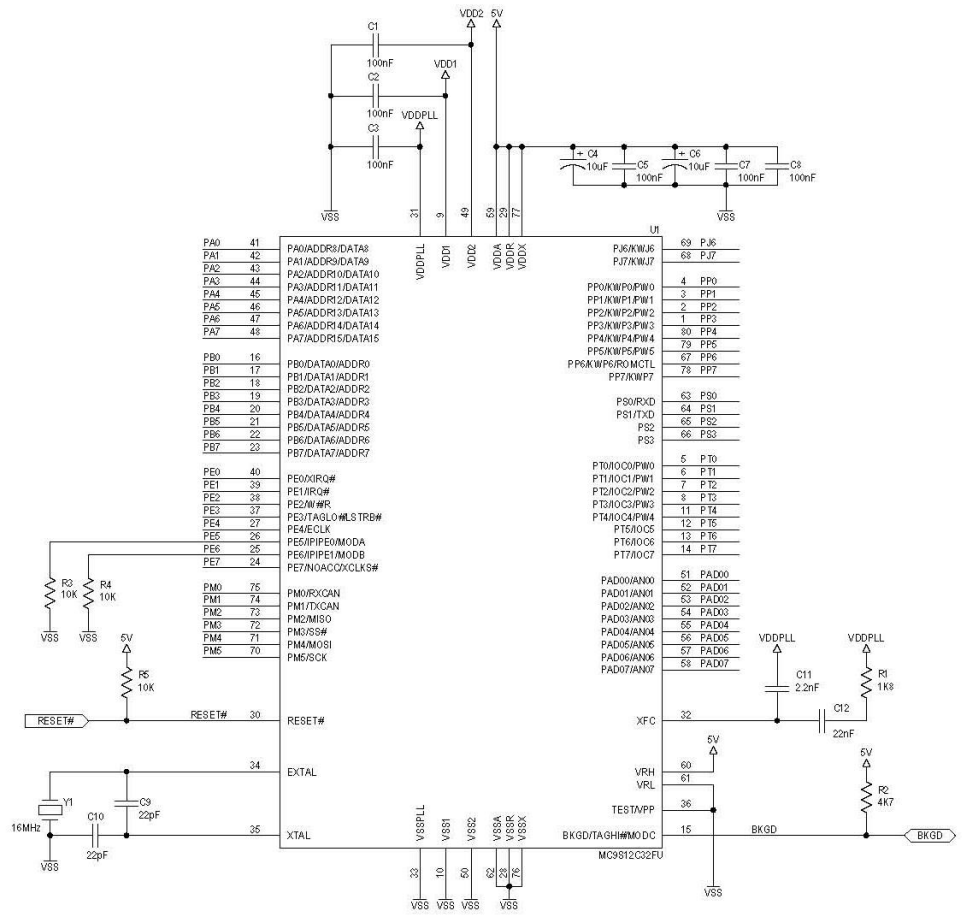
www.Asiadhe.com

Appendice F

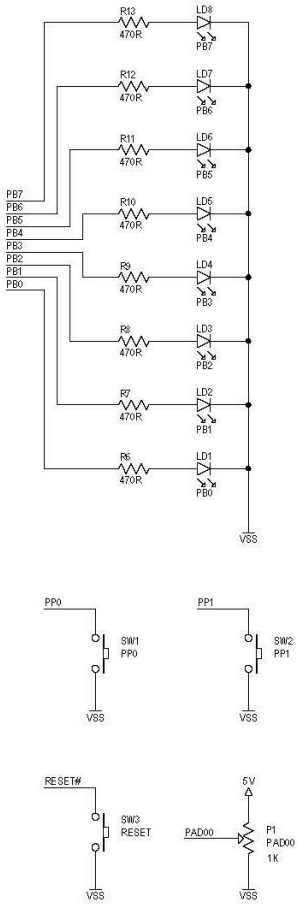
Circuits électriques de la carte HCS12 STARTER KIT



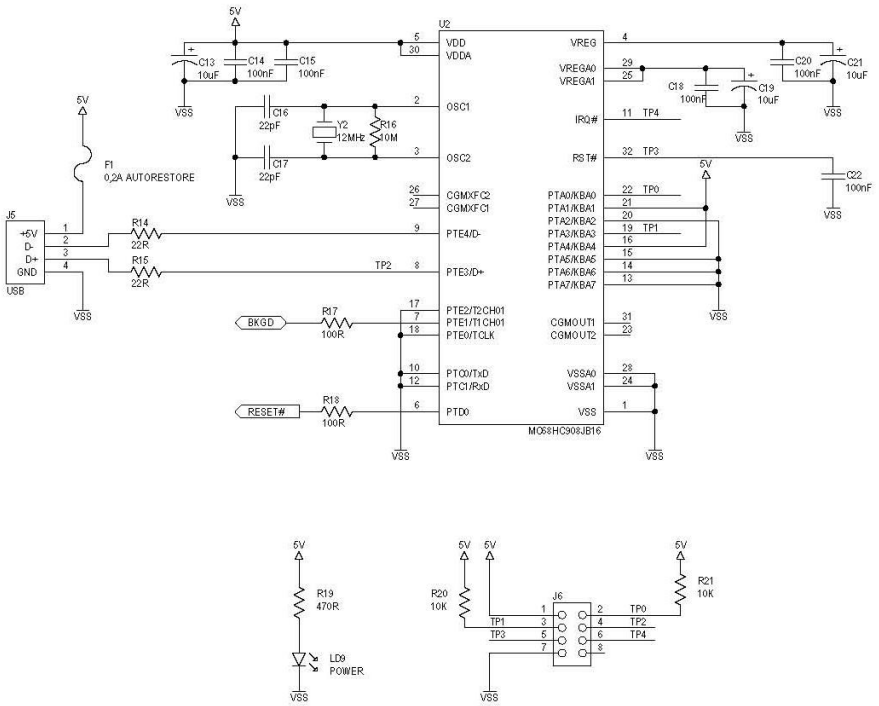
SIGNAL PINS



MCU



DEMO



USB TO BDM INTERFACE